hyväksymispäivä arvosana

arvostelija

# Vanhan järjestelmän vaiheittainen korvaaminen – ohjelmistoarkkitehtuurin soveltuvuuden arviointi

Jussi Iinatti

29.3.2012

HELSINGIN YLIOPISTO

Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET

| Tiedekunta/Osasto - Fakultet/Sektion | Laitos - Institution |
|---|---|
| Matemaattis-luonnontieteellinen tiedekunta | Tietojenkäsittelytieteen laitos |

Tekijä - Författare

Jussi Iinatti

Työn nimi - Arbetets titel

Vanhan järjestelmän vaiheittainen korvaaminen – ohjelmistoarkkitehtuurin soveltuvuuden arviointi

Oppiaine - Läroämne

Tietojenkäsittelytiede

| Työn laji - Arbetets art | Aika - Datum | Sivumäärä - Sidoantal |
|---|---|---|
| Pro gradu -tutkielma | 29.3.2012 | 63 |


Tiivistelmä – Referat

Tutkielmassa arvioidaan tapaustutkimuksen avulla ohjelmistoarkkitehtuurin soveltuvuutta vanhan järjestelmän vaiheittaiseen korvaamiseen. Esimerkkitapauksena on eräässä ohjelmistoyrityksessä toteutettu vanhan ohjelmistojärjestelmän modernisointi. Ohjelmistoarkkitehtuurin soveltuvuuden arviointiin käytetään skenaariopohjaista ATAM -analysointimenetelmää.

Tutkielmassa selvitetään, antaako suunniteltu arkkitehtuuri riittävän hyvät mahdollisuudet laadullisten vaatimusten toteutumiselle ja voidaanko arkkitehtuuria arvioimalla tukea modernisointiprojektin liiketoiminnallisten tavoitteiden saavuttamista.


ACM Computing Classification System (CCS):

D.2.1 [Requirements/Specifications], D.2.9 [Management] - *Software quality assurance (SQA)*,

D.2.11 [Software Architectures] - *Patterns*, K.6.4 [System Management] - *Quality assurance*


Avainsanat – Nyckelord

Vanhan järjestelmän modernisointi, ohjelmistoarkkitehtuuri, laatutekijä, laatuvaatimus.

Säilytyspaikka - Förvaringställe

Kumpulan tiedekirjasto, sarjanumero C-2012-

Muita tietoja

# Sisältö

## Liitteet

# 1 Johdanto

Vanha järjestelmä (*perinnejärjestelmä, a legacy system*) on informaatiojärjestelmä, jolla on käyttäjälleen liiketoiminnallista arvoa, mutta jonka muuttaminen ja kehittäminen vastaamaan uusia ja jatkuvasti muuttuvia liiketoiminnallisia vaatimuksia on merkittävän vaikeaa [BrS95]. Muutosten vaikeus voi johtua paitsi vanhan järjestelmän rakenteesta, myös siitä, että järjestelmän suunnitteluun tai toteutukseen liittyvät yksityiskohdat eivät enää ole järjestelmän kehittäjien tiedossa. Vanha järjestelmä ja järjestelmän käsittelemä tieto ovat kuitenkin tärkeää omaisuutta järjestelmää käyttäville yrityksille [BCM03]. Usein vanha järjestelmä muodostaa yrityksen informaationhallinnan selkärangan ja on tärkein väline yrityksen liiketoimintatiedon hallintaan. Vanhan järjestelmän toimintaan liittyvät ongelmat voivat aiheuttaa merkittäviä ongelmia yrityksen liiketoiminnalle [BLW97].

Kun yrityksen liiketoiminnalle tärkeä vanha järjestelmä ei enää riittävän tehokkaasti tue liiketoiminnallisten tavoitteiden saavuttamista, joudutaan järjestelmän laadulliset vaatimukset määrittelemään uudestaan. Liiketoiminnan riskitön jatkuminen edellyttää vanhan järjestelmän modernisointia uusien laadullisten vaatimusten mukaiseksi.

Ohjelmiston arkkitehtuuri määrittää rajat toteutettavan järjestelmän ohjelmistotekniselle laadulle [BCK03, DoN02]. Arkkitehtuurin arvioinnilla voidaan määrittää kuinka hyvin ohjelmiston arkkitehtuuri tai arkkitehtuuriin sisältyvät suunnittelupäätökset tukevat järjestelmälle asetettuja laadullisia vaatimuksia. Arviointi auttaa myös tunnistamaan mahdolliset laatutavoitteiden väliset ristiriidat [RoG08]. Vanhan järjestelmän modernisoinnissa tavoitteena olevan laadun parantumisen todennäköisyyttä voidaan arvioida analysoimalla järjestelmälle suunniteltua uutta arkkitehtuuria.

Tämä tutkielma on tapaustutkimus, jossa arvioidaan ohjelmistoarkkitehtuurin soveltuvuutta sellaisen ohjelmistojärjestelmän toteuttamiseen, joka tulee vaiheittain korvaamaan käytössä olevan vanhan järjestelmän. Tutkielmassa esimerkkitapauksena on projekti, jonka aikana modernisoitiin erään ohjelmistoalan yrityksen kehittämä ja ylläpitämä ohjelmistojärjestelmä. Modernisointiprojektin tavoitteena oli parantaa järjestelmän laatua sekä yrityksen sisäisiä työskentelyprosesseja.

Tutkielman tavoitteena on selvittää, ovatko arkkitehtuurisuunnitelman sisältämät arkkitehtoniset ratkaisut perusteltavissa vanhan järjestelmän modernisointiin liittyvillä laadullisilla vaatimuksilla ja antaako suunniteltu arkkitehtuuri riittävän hyvät

mahdollisuudet kehitettävälle ohjelmistolle asetettujen laatutavoitteiden saavuttamiseksi. Arkkitehtuurin arvioinnissa sovelletaan skenaariopohjaista analysointimenetelmää *Architecture Tradeoff Analysis Method* (ATAM) [KKB98, KKC00, CKK01]. Arvioinnin kohteena on uuden, korvaavan järjestelmän suunniteltu arkkitehtuuri. Vanhan järjestelmän arkkitehtuuri ja siihen modernisointiprojektin yhteydessä mahdollisesti tehtävät muutokset on jätetty arvioinnin ulkopuolelle. Arkkitehtuurin arviointiin osallistui vanhaa järjestelmää kehittävän ja ylläpitävän yrityksen henkilökuntaa, mukaan lukien johdon edustaja, vanhan järjestelmän päävastuulliset suunnittelijat sekä korvaavan järjestelmän suunnitteluun ja toteutukseen osallistuneita henkilöitä. Arviointiprosessin tuloksia tulkitaan tutkielmassa objektiivisesti.

Tutkielman kannalta merkittäviä käsitteitä esitellään luvussa 2, jossa myös määritellään arkkitehtuurin arvioinnissa käytettävät laatutekijät. Arviointimenetelmä ATAM sekä menetelmän tavoitteet ja tärkeimmät käsitteet kuvataan luvussa 3. Luvussa 4 käsitellään vanhan ohjelmistojärjestelmän modernisointia, modernisointiin käytettäviä menetelmiä, sekä modernisointiin johtavia tekijöitä. Esimerkkitapauksena oleva vanhan järjestelmän modernisointiprojekti sekä modernisoinnin kohteena oleva ohjelmistojärjestelmä esitellään luvussa 5. Modernisointiprojektissa tavoiteltuun järjestelmän laadulliseen kehittymiseen liittyvät vaatimukset on kuvattu luvussa 6. Luvussa 7 esitellään arvioinnin kohteena oleva, korvaavan järjestelmän ohjelmistoarkkitehtuuri. Arkkitehtuuri-suunnitelmaan sisältyvät suunnittelupäätökset analysoidaan luvussa 8, jossa myös arvioidaan arkkitehtuurin soveltuvuutta korvaavan järjestelmän toteuttamiseen.

# 2 Laatutekijät ja ohjelmistoarkkitehtuuri

## 2.1 Laadulliset vaatimukset ja laatutekijät

Ohjelmistojärjestelmän käyttäjälleen tuottama hyöty muodostuu järjestelmän tarjoamista toiminnoista ja järjestelmään liittyvistä ei-toiminnallisista tekijöistä. Ohjelmisto-arkkitehtuurien yhteydessä ei-toiminnallisista tekijöistä käytetään usein termiä *laatutekijä* [CSL09]. IEEE-standardin [IEE98] mukaan ohjelmistojärjestelmän laatu tarkoittaa sitä tasoa, jolla joukko valittuja laatutekijöitä toteutuu järjestelmässä.

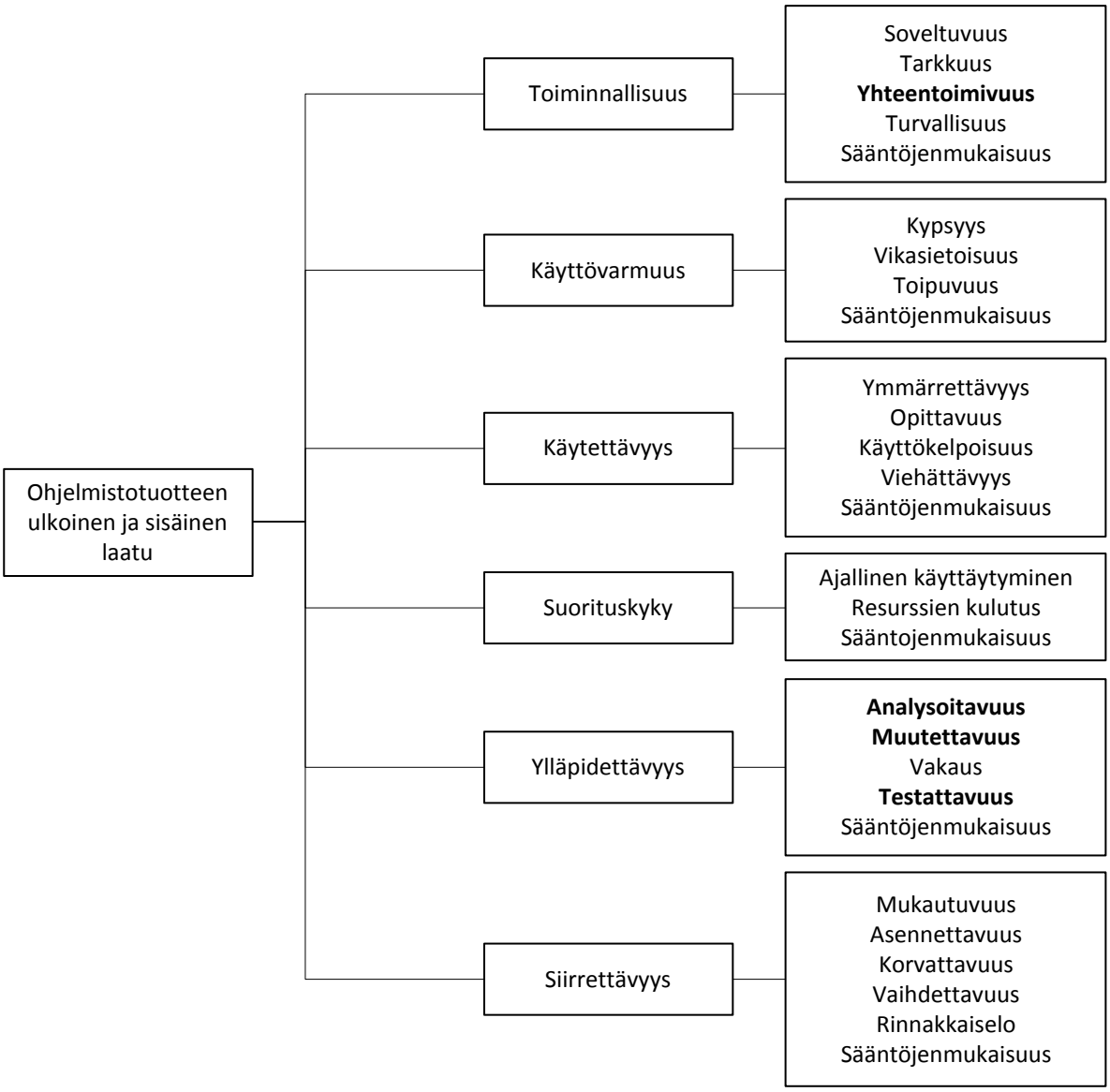
Laatutekijä on siis jokin järjestelmän ominaisuus, joka yhdessä muiden laatutekijöiden kanssa määrittää järjestelmän laadun. Yksittäisiä laatutekijöitä ovat esimerkiksi testattavuus, muutettavuus ja saavutettavuus. Kun käsitellään järjestelmän laatua, on tehtävä selväksi, minkä laatutekijöiden määrittämästä laadusta on kyse.

Käytettävästä lähteestä riippuen voi sama laatutekijä olla määritelty eri tavoin. Järjestelmän laatua arvioitaessa on tärkeää, että arvioinnissa käytettävien laatutekijöiden määrittelyt ovat yksiselitteisesti sekä laadullisen arvion tuottajien että arvion käyttäjien tiedossa.

Kun pyritään korkealaatuisen ohjelmistojärjestelmän tuottamiseen, tulisi järjestelmän vaatimusmäärittelyn sisältää paitsi kuvaus vaadituista toiminnoista, myös kuvaus eri laatutekijöihin perustuvista laadullisista vaatimuksista. Usein laadulliset vaatimukset on kuitenkin määritelty epätarkasti ja ne voivat olla keskenään ristiriitaisia. Kun laadulliset vaatimukset ovat epätarkkoja, on vaatimusten saavuttamista tukevien ominaisuuksien toteuttaminen ohjelmistoon haastavaa. Epätarkkojen vaatimusten toteutumista valmiissa ohjelmistossa on vaikea varmentaa. Saavuttamatta jääneet laadulliset vaatimukset ovat tärkeä syy ohjelmistokehitysprojektien epäonnistumiseen [MCY99].

## 2.2 Laatutekijöiden määrittely

Tutkielman esimerkkitapaukseen liittyvän korvaavan järjestelmän laadulliset vaatimukset esitellään luvussa 6. Laadullisten vaatimusten yhteydessä käytettävien laatutekijöiden määrittelyt, lukuun ottamatta laatutekijöitä *saavutettavuus* ja *ositettavuus*, ovat osa kuvassa 1 esitettyä ISO/IEC 9126-1:2001 -standardin [ISO01] mukaista ohjelmistotuotteen laatumallia. Laatumalliin kuulumaton saavutettavuus on yhdistelmä kolmesta standardin määrittelemästä laatutekijästä: kypsyys, vikasietoisuus ja toipuvuus. Ositettavuus puolestaan on määritelty ATAM-menetelmää käsittelevässä teoksessa [CKK01].




*Kuva 1: ISO 9126-1-standardin mukainen ohjelmistotuotteen laatumalli [ISO01]*

Laadullisten vaatimusten määrittelyssä käytettävät laatutekijät näkyvät kuvassa 1 vahvennettuina ja niiden määrittelyt ovat seuraavissa luvuissa. Laatutekijöiden määrittelyt ovat esimerkkitapauksen modernisointiprojektissa määriteltyjen tavoitteiden mukaisessa tärkeysjärjestyksessä.

Laatumallin mukaisessa hierarkkisessa rakenteessa keskimmäisellä tasolla määritelty *ylläpidettävyys* (maintainability) on esimerkkitapauksen tavoitteiden kannalta erittäin tärkeässä asemassa oleva laatutekijä. Ylläpidettävyys mittaa, kuinka helposti ohjelmistotuotetta on mahdollista muuttaa. Muutokset voivat olla esimerkiksi korjauksia,

parannuksia tai tuotteen mukauttamista muutoksiin, jotka kohdistuvat tuotteen ajoympäristöön, laatuvaatimuksiin tai vaadittuun toiminnallisuuteen [ISO01]. Sen katsotaan koostuvan kolmesta osatekijästä: testattavuus, muutettavuus ja analysoitavuus. Ylläpidettävyyden toteutumista korvaavassa järjestelmässä arvioidaan sillä perusteella, kuinka nämä osatekijät toteutuvat.

### 2.2.1 Testattavuus

*Testattavuudella* (testability) tarkoitetaan, kuinka helppoa tai työlästä on varmentaa järjestelmän toiminnan oikeellisuus siihen tehtyjen muutosten jälkeen [ISO01].

Esimerkkitapauksessa järjestelmän testattavuuden arviointi perustuu siihen, kuinka helposti järjestelmään syntyneitä vikoja voidaan havaita toteuttamalla ja suorittamalla erilaisia testejä. Arviointi on jaettu järjestelmälle ominaisten testien tehokkuuden arviointiin ja testien toteuttamisen kustannusten arviointiin. Testien tehokkuudella tarkoitetaan todennäköisyyttä, jolla järjestelmän sisältämä vika havaitaan testien suorittamisen yhteydessä. Vikojen havaitsemisen todennäköisyyden on katsottu olevan sitä suurempi, mitä kattavammin järjestelmän sisältämää ohjelmakoodia ja eri toimintoja on mahdollista suorittaa testien avulla. Järjestelmälle ominaisilla testeillä tarkoitetaan testejä, joiden toteuttamista ja suorittamista järjestelmän rakenne ja toteutusteknologiat parhaiten tukevat. Kustannusten arvioinnin mittarina on käytetty testien toteuttamiseen kuluvaa aikaa.

### 2.2.2 Muutettavuus

*Muutettavuudella* (changeability) tarkoitetaan, kuinka helppoa tai työlästä on toteuttaa järjestelmään jokin määritelty muutos [ISO01].

Esimerkkitapauksessa keskitytään arvioimaan järjestelmään tehtävän muutoksen aiheuttamaa kustannusta. Kustannus on usein helpointa mitata ajassa, mutta mittarina voidaan käyttää myös muutosten oletettuja heijastusvaikutuksia. Voidaan esimerkiksi arvioida niiden komponenttien lukumäärää, joiden muokkaamista ei voida välttää, kun järjestelmään toteutetaan määritelty muutos.

### 2.2.3 Analysoitavuus

*Analysoitavuudella* (analysability) tarkoitetaan, kuinka helppoa tai työlästä on tunnistaa järjestelmästä jonkin havaitun vian aiheuttaja tai ne järjestelmän osat, joihin muutokset täytyy kohdistaa tietyn ominaisuuden toteuttamiseksi [ISO01]. Analysoitavuus mittaa

kuinka helppoa järjestelmän teknisen toteutuksen ymmärtäminen tai omaksuminen on kehittäjän näkökulmasta.

### 2.2.4 Ositettavuus

*Ositettavuus* (subsetability) on ohjelmistojärjestelmän ominaisuus, joka tukee järjestelmän yhden osakokonaisuuden toteuttamista riippumatta järjestelmän muiden osien toteutuksen ajankohdasta tai toteutusjärjestyksestä. Ositettavuus mahdollistaa vaiheittaisen ohjelmistokehitysmallin, jossa ohjelmistosta toteutetaan yksi kerrallaan mahdollisimman pieniä, suoritettavissa olevia osia [CKK01].

Ositettavuus ei ole ISO-standardeissa määritelty laatutekijä. Standardin ISO/IEC 9126-1:2001 [ISO01] määrittelemät laatutekijät muutettavuus ja testattavuus sekä erityisesti standardin ISO/IEC 25010:2011 [ISO11] määrittelemä modulaarisuus tukevat toteutuessaan järjestelmän ositettavuutta. Ositettavuus voitaisiin määrittelynsä puolesta sijoittaa ISO-standardin ylläpidettävyyden osatekijäksi.

### 2.2.5 Yhteentoimivuus

*Yhteentoimivuudella* (interoperability) tarkoitetaan, kuinka hyvin järjestelmä toimii yhteistyössä yhden tai useamman ulkoisen järjestelmän kanssa.

Esimerkkitapauksessa arvioidaan erityisesti sitä, kuinka helppoa järjestelmä on liittää vanhan järjestelmän käyttämiin ulkoisiin järjestelmiin. Ulkoiset järjestelmät voivat toimia joko järjestelmän palvelujen käyttäjänä tai tarjota palveluja järjestelmälle.

### 2.2.6 Saavutettavuus

*Saavutettavuudella* (availability) tarkoitetaan järjestelmän kykyä säilyä sellaisessa tilassa, jossa se pystyy suorittamaan vaaditun toimenpiteen määriteltynä ajankohtana. Järjestelmän ulkopuolelta tarkasteltuna saavutettavuus tarkoittaa aikaa, jolloin järjestelmä on käyttäjiensä käytettävissä [ISO01]. Saavutettavuuteen vaikuttavat virheiden esiintymistiheys, järjestelmän kyky säilyttää toiminnallisuutensa virhetilanteissa sekä virheiden aiheuttamien käyttökatkojen kesto. Käyttökatkon kestoon vaikuttaa myös virheen korjaamisesta vastaavan osapuolen informoimiseen kuluva aika. Standardi ISO/IEC 9126-1:2001 [ISO01] ei määrittele saavutettavuutta itsenäisenä laatutekijänä. Standardin mukaan saavutettavuus on yhdistelmä laatutekijöistä kypsyys, vikasietoisuus ja toipuvuus.

Bass et. al [BCK03] määrittelee saavutettavuudelle yleisesti käytetyn kaavan

$$A = \frac{E[\text{Uptime}]}{E[\text{Uptime}] + E[\text{Downtime}]},$$

jossa **A** on saavutettavuudelle laskettava arvo prosentteina, **E[Uptime]** on arvio kahden vian esiintymisen välillä kuluvasta ajasta tunteina (mean time between failure) ja **E[Downtime]** on arvio vian korjaamiseen keskimäärin kuluvasta ajasta tunteina (mean time to recovery).

## 2.3 Ohjelmistoarkkitehtuuri

IEEE-standardin [IEE00] mukaan ohjelmistoarkkitehtuuri on ohjelmistojärjestelmän perusrakenne, joka kuvaa järjestelmän sisältämät osat, osien väliset keskinäiset suhteet ja osien suhteet ympäristöönsä. Lisäksi arkkitehtuuri sisältää periaatteet, joita ohjelmistojärjestelmän suunnittelussa ja kehityksessä tulee noudattaa.

Tässä tutkielmassa ohjelmistoarkkitehtuuria käsitellään preskriptiivisenä mallina, joka on olemassa ennen ohjelmiston toteuttamista. Arkkitehtuuri on korkean abstraktiotason kuvaus, jonka tarkoituksena on helpottaa ohjelmiston kompleksisuuden hallintaa suunnittelu- ja toteutusvaiheessa. Arkkitehtuuridokumentaation tulee kuvata ohjelmiston jako pääosiin, pääosien väliset suhteet, pääosien väliset kommunikointikanavat sekä pääosien sijoittelu laitteistotasolla.

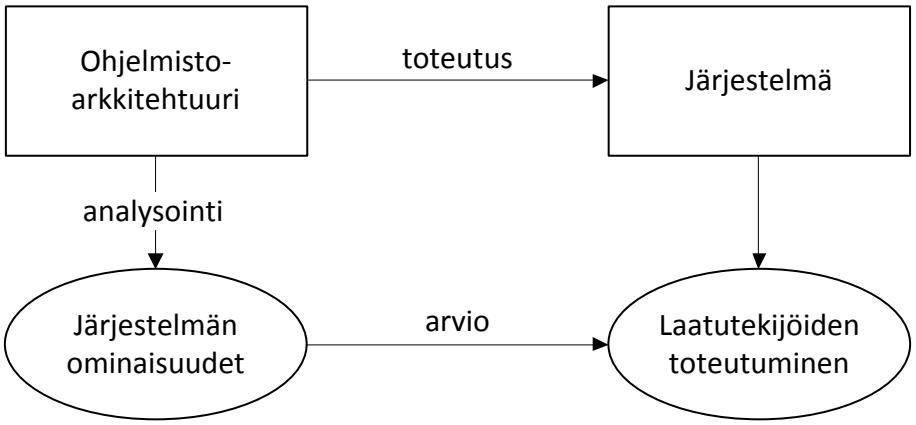
Tarjoamalla abstraktin kuvauksen järjestelmästä arkkitehtuuri paljastaa tietyt järjestelmän ominaisuudet piilottaen samalla muita ominaisuuksia. Kuvauksen perusteella järjestelmän suunnittelijat voivat päätellä, kuinka hyvin järjestelmä tulee vastaamaan sille asetettuja vaatimuksia. Arkkitehtuurikuvaus mahdollistaa järjestelmän analysoinnin annettujen laatutekijöiden suhteen ennen järjestelmän toteuttamista [Gar00]. Sekä tieteellisissä että teollisissa yhteisöissä on havaittu, että ohjelmistoarkkitehtuuri määrittää rajat sitä noudattavan järjestelmän ohjelmistotekniselle laadulle [BCK03, BZJ04, DoN02].

## 2.4 Ohjelmistoarkkitehtuurin arviointi

Ohjelmistoarkkitehtuurin arvioinnin pääasiallinen tavoite on selvittää minkälaiset mahdollisuudet arkkitehtuuri antaa sellaisen järjestelmän toteuttamiselle, joka pystyy täyttämään sille asetetut laadulliset vaatimukset [BZJ04]. Arvioinnilla pyritään myös löytämään arkkitehtuurin mahdollisesti sisältämät riskit, jotka toteutuessaan vaikuttaisivat negatiivisesti valittujen laatutekijöiden toteutumiseen. Ohjelmistoarkkitehtuurin arviointiin on käytettävissä useita eri analysointimenetelmiä, joiden avulla pyritään

ennustamaan järjestelmän laatua ennen sen toteuttamista. Arkkitehtuurin analysoinnilla ei pyritä tuottamaan tarkkoja arvioita järjestelmän tulevasta laadusta, vaan muodostamaan käsitys arkkitehtuurin pääasiallisista vaikutuksista järjestelmän laatuun [DoN02].

Ohjelmistoarkkitehtuurin noudattaminen tuottaa järjestelmään joukon ominaisuuksia, jotka voidaan selvittää arkkitehtuurisuunnitelmaa analysoimalla. Ominaisuuksien perusteella voidaan arvioida valittujen laatutekijöiden toteutumista valmiissa järjestelmässä. Ohjelmistoarkkitehtuurin analysoinnin pääkomponentit ja niiden väliset suhteet on esitetty kuvassa 2.




*Kuva 2: Ohjelmistoarkkitehtuurin analysointi [BLB04]*

Arkkitehtuuria analysoimalla ei pystytä arvioimaan kaikkia järjestelmän kokonaislaatuun vaikuttavia tekijöitä [CKK01]. Esimerkkinä voidaan käyttää laatutekijää *käytettävyys*, joka kuvaa käyttäjän kykyä hyödyntää järjestelmää tehokkaasti. Arkkitehtuuritason ratkaisuilla voi olla suurikin vaikutus käytettävyyteen esimerkiksi järjestelmän suorituskyvyn kautta. Käytettävyyden arviointi pelkästään arkkitehtuuria analysoimalla ei kuitenkaan voi tuottaa tarkkaa ennustetta laatutekijän toteutumisesta, sillä monet käytettävyyteen vaikuttavat järjestelmän yksityiskohdat jäävät tarkkuustasonsa vuoksi kokonaan arkkitehtuurikuvauksen ja siten arkkitehtuurin analysoinnin ulkopuolelle.

Ohjelmistoarkkitehtuuri arvioidaan yleensä heti arkkitehtuurisuunnitelman valmistuttua, ennen ohjelmiston toteutusta [BaG04, CKK01]. Arviointi perustuu suunnitelmaan sisältyvien arkkitehtonisten päätösten vaikutusten analysointiin. Arviointi voidaan tehdä, vaikka arkkitehtuurisuunnitelma ei olisi kattava. Tällöin analysoinnin kohteena ovat voimassa olevien päätösten lisäksi myös harkinnassa olevat arkkitehtoniset ratkaisut [CKK01]. Koska ohjelmiston arkkitehtuurilla on merkittävä vaikutus valittujen

laatutekijöiden toteutumiseen, on tärkeää, että arkkitehtuuri arvioidaan ainakin tärkeimmiksi luokiteltujen laatutavoitteiden osalta mahdollisimman aikaisessa vaiheessa ohjelmiston kehitysprosessia. On nopeampaa sekä edullisempaa havaita ja korjata suunnitelmissa olevia virheitä ohjelmistokehitysprosessin alkuvaiheessa. Tehokasta ohjelmistoarkkitehtuurin analysointimenetelmää soveltamalla voidaan saavuttaa merkittävää liiketoiminnallista hyötyä [BZJ04]. Ohjelmistoarkkitehtuurin arviointi on kustannustehokas menetelmä ohjelmistoprojektin riskienhallintaan ja järjestelmän laadun ylläpitämiseen [KCW00].

## 2.5 Skenaariopohjaiset analysointimenetelmät

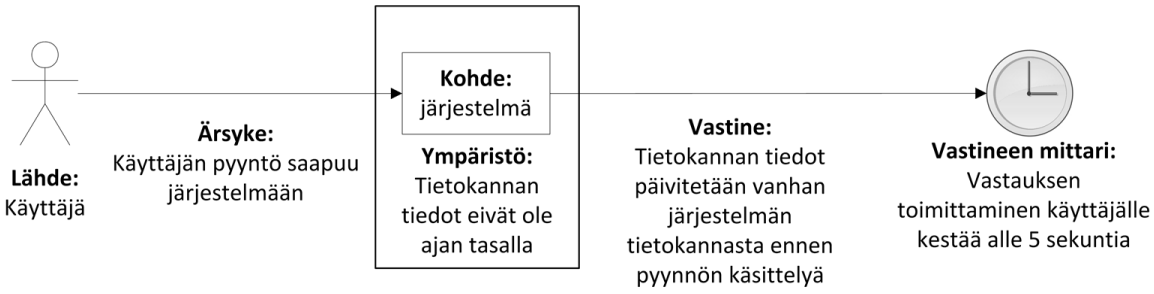
Ohjelmistoarkkitehtuurin arvioinnilla pyritään selvittämään toteutettavan järjestelmän laatu valittujen laatutekijöiden suhteen. Laatutekijöitä sellaisenaan ei kuitenkaan voida käyttää järjestelmän laadullisten vaatimusten määrittelyyn. Esimerkkinä voidaan käyttää laatutekijää *muutettavuus*, joka kuvaa järjestelmään tehtävästä muutoksesta aiheutuvaa kustannusta. Jos järjestelmän laatuvaatimuksena on ”hyvä muutettavuus”, ei järjestelmän suunnittelussa voida juurikaan vaikuttaa laatutekijän toteutumiseen. Muutoksen kustannusvaikutusten minimoimiseen tähtäävä arkkitehtuuritason ratkaisu riippuu voimakkaasti oletetun muutoksen luonteesta, mutta vaatimus ei kerro minkälaisia muutoksia järjestelmän toteutuksen tulisi mahdollistaa tai helpottaa.

Sen sijaan, että laatutekijöiden määrittelyjä tarkennettaisiin, voidaan laadullisia vaatimuksia kuvata tarkemmin sitomalla vaatimukseen liittyvä laatutekijä rajattuun kontekstiin. Konteksti voidaan kuvata *skenaariolla*, joka sisältää laatutekijän lisäksi informaatiota siitä käyttötilanteesta, jossa laatutekijän on toteutettava asetetut raja-arvot. Myös raja-arvot voidaan kuvata osana skenaariota.

Kazman et al. [KCW00] määrittelee skenaarion ”yksittäiseksi, järjestelmän ja sen sidosryhmän edustajan väliseksi vuorovaikutteiseksi tapahtumaksi”. Skenaario muistuttaa oliopohjaisen ohjelmistosuunnittelun käsitettä *käyttötapaus*. Käyttötapauksissa keskitytään kuitenkin järjestelmän suorituksen aikana tapahtuvaan järjestelmän ja sen käyttäjän väliseen vuorovaikutukseen. Skenaario voi kuvata myös muun tyyppisiä vuorovaikutustilanteita. Esimerkiksi ohjelmoija tai järjestelmän ylläpitäjä voi muuttaa järjestelmän sisäistä toteutusta käyttäen muutoksen toteuttamiseen muita menetelmiä kuin tavalliselle käyttäjälle näkyvää käyttöliittymää.

Skenaario kuvaa järjestelmälle asetettua laadullista vaatimusta konkreettisen esimerkin avulla. Konkreettinen esimerkki auttaa arviointiin osallistuvia henkilöitä ymmärtämään

samalla tavalla, mitä tietyllä laatutekijällä tarkoitetaan. Kuvassa 3 oleva ATAM-menetelmän mukainen skenaario antaa esimerkin yhteentoimivuuteen liittyvästä vaatimuksesta. Skenaariopohjaisia menetelmiä pidetään kypsäksi kehittyneenä ohjelmistoarkkitehtuurin arviointimenetelmien luokkana [BaG04] ja niiden on arvioitu olevan kustannustehokkaita työkaluja arkkitehtuurisuunnittelun riskien hallintaan [KCW00].




*Kuva 3: ATAM-menetelmän mukainen skenaario kaaviomuotoisena*

Babar et al. ovat vertailleet joukkoa yleisimmin käytettyjä skenaariopohjaisia arkkitehtuurin analysointimenetelmiä [BaG04, BZJ04]. Tutkituilla menetelmillä on sama tavoite: arvioon perustuvan, arkkitehtuuritason laadullisen määrittelyn tuottaminen järjestelmälle. Menetelmät poikkeavat toisistaan siinä, kuinka arkkitehtuuritason laatua tarkastellaan ja mistä komponenteista arkkitehtuuritason laadun arvio koostuu. SAAM [KBW94] painottaa arkkitehtuuriin sisältyvien riskien tunnistamista. ALMA [BLB04] keskittyy yhden laatutekijän - muutettavuuden - arviointiin eri näkökulmista. PASA [WiS02] on ensisijaisesti menetelmä järjestelmän suorituskykyyn liittyvien riskien tunnistamiseen ja lieventämiseen. ATAM-menetelmän avulla tunnistetaan ja analysoidaan arkkitehtuurin sisältämiä *herkkyys*- ja *kompromissipisteitä*, joihin sisältyvät ratkaisut voivat estää tavoiteltujen laatutekijöiden toteutumisen halutulla tasolla [CKK01, KKC00, KKB98].

Yksi merkittävimpiä eroja menetelmien välillä on niiden käsittelemien laatutekijöiden lukumäärä [BaG04]. Edellä mainituista menetelmistä pelkästään ATAM arvioi järjestelmän laatua useiden laatutekijöiden suhteen, muiden keskittyessä yhteen laatutekijään. ATAM-menetelmää käytettäessä pyritään löytämään ja analysoimaan erityisesti ne suunnittelupäätökset, jotka vaikuttavat yhteen tai useampaan valituista laatutekijöistä.

SAAM- ja ATAM-menetelmien mukaiseen arviointiin pyritään ottamaan mukaan kaikki järjestelmän tärkeimmät sidosryhmät kuten ohjelmistoarkkitehdit,

ohjelmistosuunnittelijat, loppukäyttäjät sekä järjestelmää kehittävän ja järjestelmää käyttävän yrityksen päätöksentekijät [CKK01]. Muut edellä mainituista menetelmistä ottavat arviointiin mukaan vain järjestelmän suunnittelijoita ja/tai toteuttajia [BaG04].

# 3 Architecture Tradeoff Analysis Method (ATAM)

## 3.1 Taustaa

Architecture Tradeoff Analysis Method (ATAM) on Carnegie Mellon yliopiston SEI-instituutin kehittämä skenaariopohjainen ohjelmistoarkkitehtuurien analysointimenetelmä [KKB98, KKC00, CKK01]. Monista muista analysointimenetelmistä poiketen ATAM sisältää yksityiskohtaisen kuvauksen, kuinka menetelmää käytännön tilanteessa sovelletaan ja minkälaisia tehtäviä analysointiprosessi sisältää [BZJ04]. ATAM on kehitetty SAAM-menetelmän pohjalta ja menetelmien kehittämiseen on osallistunut samoja henkilöitä [CKK01]. ATAM soveltuu käytettäväksi missä tahansa ohjelmistokehitysprosessin vaiheessa, mutta menetelmä on tehokkaimmillaan kun arvioinnin kohteena on arkkitehtuurisuunnitelman lopullinen versio [BaG04].

Babar et al. ovat artikkelissaan [BZJ04] jakaneet arkkitehtuurien arviointimenetelmiä neljään luokkaan niiden kypsyysasteen mukaan: *alustava* (inception), *kehitteillä* (development), *jalostunut* (refinement) ja *lepotilassa oleva* (dormant). Heidän mukaansa ATAM on jalostunut arviointimenetelmä. ATAM-menetelmää on käytetty useilla eri sovellusalueilla ja sitä on jatkuvasti kehitetty saatujen kokemusten perusteella. Clements et. al [CKK01] sekä Bass et. al [BCK03] kuvaavat kirjoissaan useita käytännön esimerkkitapauksia menetelmän käytöstä.

## 3.2 Arviointiin osallistujat

Valmis ohjelmistojärjestelmä palvelee laajaa sidosryhmien joukkoa. Skenaariopohjaisissa analysointimenetelmissä sidosryhmien merkitys on kriittinen, koska sidosryhmien edustajat ovat päävastuussa arvioinnin perustana olevien skenaarioiden luomisesta [BZJ04]. Useat ohjelmistoprojektin sidosryhmät ovat kiinnostuneita paitsi varsinaisesta lopputuotteesta, myös ohjelmistoprojektin aikana syntyvistä muista tuloksista. Jotta ohjelmistoarkkitehtuurin analysointi onnistuisi, on tärkeää tunnistaa kehitettävän järjestelmän tärkeimmät sidosryhmät sekä näiden sidosryhmien odotukset projektin ja valmiin järjestelmän suhteen [BZJ04]. ATAM edellyttää sidosryhmien aktiivista osallistumista arviointiin laadukkaan lopputuloksen saavuttamiseksi [CKK01].

ATAM-menetelmän mukainen analysointiprosessi jakautuu kahteen vaiheeseen, jotka sisältävät osittain samoja tehtäviä. Ensimmäiseen vaiheeseen osallistuu pelkästään arkkitehtuurianalysointia varten koottu arviointiryhmä. Clements et al. [CKK01] kuvaa yhteensä kahdeksan roolia, jotka arviointiryhmässä tarvitaan. Sama arviointiryhmän jäsen voi toimia useammassa kuin yhdessä roolissa, ja pienimmäksi toimivaksi arviointiryhmän kooksi on ilmoitettu neljä henkilöä.

Toiseen vaiheeseen osallistuu ensimmäisen vaiheen arviointiryhmä kokonaisuudessaan sekä lisäksi järjestelmän tärkeimpien sidosryhmien edustajia. Clements et al. [CKK01] kuvaa sidosryhmien edustajille 20 mahdollista roolia, ryhmiteltynä järjestelmän toteuttajiin, järjestelmän käyttäjiin, järjestelmän ylläpitäjiin ja järjestelmän kanssa välillisesti tekemisissä oleviin. ATAM ei anna tarkkaa lukumäärää toisessa vaiheessa tarvittaville sidosryhmien edustajille.

## 3.3 Tärkeimmät käsitteet

Seuraavissa luvuissa kuvataan ATAM-menetelmän tärkeimmät käsitteet. Menetelmässä käytetään käsitteitä, joiden merkitys ei ole vakiintunut samaksi muissa yhteyksissä käytettynä.

### 3.3.1 Arkkitehtoninen lähestymistapa

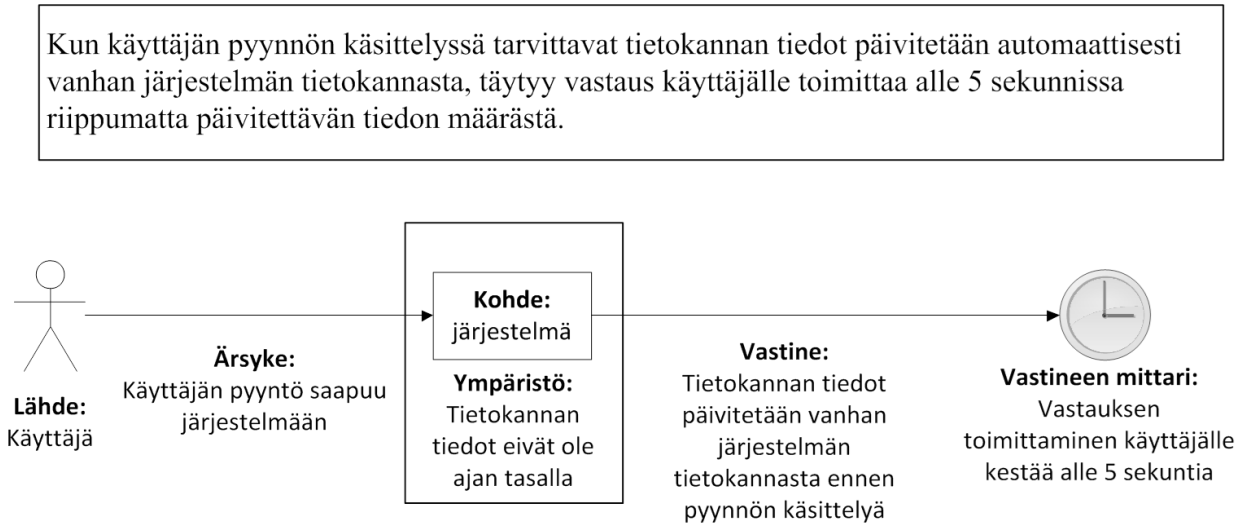
Useiden ohjelmistojärjestelmien toteutuksen yhteydessä on havaittu, että järjestelmien arkkitehtuureissa on tunnistettavissa toistuvia, samankaltaisia ratkaisumalleja, jotka tuottavat järjestelmiin määritellyissä olosuhteissa poikkeuksellisen hyviä ominaisuuksia [TMD10]. Tällaisia tunnistettavia ja toistettavissa olevia arkkitehtuuritason ratkaisumalleja on pyritty luokittelemaan eri tavoin. Kaksi usein käytettyä ratkaisumallien luokkaa ovat arkkitehtoninen tyyli ja arkkitehtoninen suunnittelumalli. Taylor et al. [TMD10] kuitenkin toteaa, että tietyn arkkitehtonisen ratkaisun sijoittaminen jompaankumpaan näistä luokista on usein tulkinnanvarainen asia.

Arkkitehtuuritason ratkaisumalleja käsittelevä luokittelu ja terminologia eivät ole vakiintuneet ohjelmistoteollisuudessa. Usein termien merkitys ymmärretään eri tavoin kuin on tarkoitettu [CKK01]. Virheellisten tulkintojen välttämiseksi ATAM-menetelmän yhteydessä käytetään väljästi määriteltyä termiä *arkkitehtoninen lähestymistapa*, joka voi tarkoittaa mitä tahansa tunnistettavissa ja dokumentoitavissa olevaa arkkitehtuuritason suunnittelupäätöstä tai suunnittelupäätösten joukkoa.

### 3.3.2 Skenaario

Skenaarioiden avulla tuodaan tarkasti esiin ne laadulliset vaatimukset, joiden toteutumisen mahdollisuuksia arkkitehtuurissa arvioidaan [CKK01]. ATAM luokittelee skenaariot kolmeen ryhmään: käyttötapausskenaariot, kasvuskenaariot ja kokeelliset skenaariot. Käyttötapausskenaariot kuvaavat käyttäjän ja valmiin järjestelmän suunniteltua vuorovaikutusta. Kasvuskenaariot kuvaavat odotettavissa olevia, järjestelmään kohdistuvia muutostarpeita. Kokeellisten skenaarioiden avulla pyritään löytämään järjestelmän suunnitellusta rakenteesta ja toteutustavasta johtuvia tekijöitä, jotka asettavat rajoituksia järjestelmän muuttamiselle tai jatkokehitykselle.

Kaikki skenaariot pyritään kuvaamaan samaa rakennetta noudattaen. Kuvassa 4 on esitetty ATAM-menetelmän mukainen käyttötapausskenaario sekä tekstinä että määrämuotoisena kaaviona.



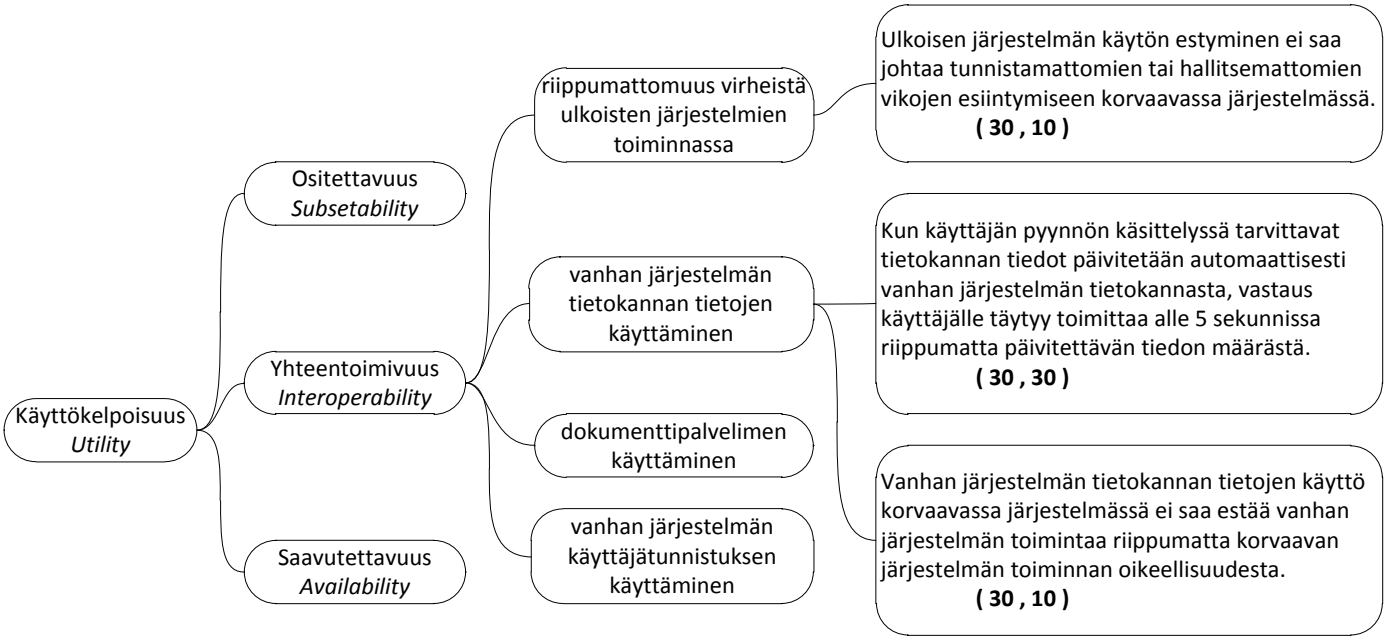

*Kuva 4: ATAM-menetelmän mukainen käyttötapausskenaario tekstinä ja kaaviona*

Skenaarion tulisi sisältää kolme osaa: *ärsyke, ympäristö* ja *vastine.* Ärsyke kuvaa millainen syöte järjestelmään saapuu ja mikä on syötteen alkuperä. Ympäristö kuvaa missä tilassa järjestelmä on ärsykkeen saapuessa. Erityisesti normaalista tilanteesta poikkeavat ympäristötekijät tulisi kuvata tarkasti. Ympäristö voi myös tarkentaa ärsykkeen kohdetta, mikäli kohteena on vain tietty osa järjestelmää. Vastine kuvaa, minkälaista vastausta järjestelmältä edellytetään. Vastauksen tuottamiseen liittyvät raja-arvot ja mahdolliset mittarit kuvataan skenaarion yhteydessä.

Skenaariot mahdollistavat järjestelmän suunnitteluvaiheessa epämääräisesti kuvattujen laatuvaatimusten konkretisoinnin [CKK01]. Skenaarioihin pystytään liittämään tarkkoja laatutekijöihin liittyviä vaatimuksia ja vaatimusten toteutumisen arvioinnissa käytettäviä mittareita. Toisaalta skenaariot voidaan arvioinnin alkuvaiheessa jättää käytettävissä olevan informaation puutteiden vuoksi epätarkoiksi ja tarkentaa niitä arviointiprosessin myöhemmissä vaiheissa, sitä mukaa kun järjestelmää koskevat vaatimukset tarkentuvat.

### 3.3.3 Laatupuu

Laatupuu on puumuotoinen esitys, joka sisältää ne laadulliset vaatimukset, joiden toteutumisen mahdollisuuksia arkkitehtuurissa arvioidaan [CKK01]. Kuvassa 5 on esitetty osa ATAM-menetelmän mukaisesta laatupuusta.




*Kuva 5: Osa ATAM-menetelmän mukaisesta laatupuusta*

Jokaista laatutekijää kohti tuotetaan yksi tai useampia tarkennuksia (refinement). Tarkennuksia käyttämällä voidaan määritellä, mihin järjestelmän ominaisuuteen tai toiminnallisuuteen laadulliset vaatimukset kohdistuvat. Esimerkiksi saavutettavuuteen liittyvät laadulliset vaatimukset voidaan jakaa laitteiston häiriöitä koskeviin vaatimuksiin ja toisaalta ohjelmiston häiriöitä koskeviin vaatimuksiin. Yhteentoimivuuteen liittyvät vaatimukset voidaan rajata koskemaan vain tiettyjä kehitettävän järjestelmän kanssa vuorovaikutuksessa olevia ulkoisia järjestelmiä.

Jokaista laatutekijän tarkennusta kohti laaditaan vähintään yksi skenaario. Skenaariot järjestetään analysointiprosessin aikana tärkeysjärjestykseen kahden tekijän perusteella. Ensimmäinen tekijä kuvaa skenaarion merkitystä koko ohjelmistokehitysprojektin ja sen lopputuloksena syntyvän järjestelmän onnistumisen kannalta. Toisen tekijän avulla kuvataan, kuinka vaikeaa skenaarion mukaiset laadulliset vaatimukset täyttävä ratkaisu on toteuttaa. Analysointiin osallistuvat henkilöt antavat molemmille tekijöille arvot kolmiportaisella asteikolla. Skenaariot, joiden molemmilla tekijöillä on asteikon suurin arvo, ovat tärkeysjärjestyksessä ensimmäisenä. Arkkitehtuurin soveltuvuuden arvioinnissa kiinnitetään erityisesti huomiota niihin suunnittelupäätöksiin, jotka vaikuttavat tärkeysjärjestyksessä ensimmäisinä olevien skenaarioiden toteutumiseen.

### 3.3.4 Herkkyyspisteet ja kompromissipisteet

ATAM-menetelmän mukaisessa arkkitehtuurin arvioinnissa tunnistetaan ja nimetään osa arkkitehtuurisuunnitelman sisältämistä päätöksistä *herkkyyspisteiksi* (sensitivity point) ja *kompromissipisteiksi* (tradeoff point) [CKK01].

Herkkyyspiste on yhteen tai useampaan järjestelmän komponenttiin tai komponenttien väliseen suhteeseen liittyvä arkkitehtoninen ratkaisu, joka vaikuttaa kriittisellä tavalla tietyn laatutekijän suhteen määritellyn laadullisen vaatimuksen toteutumiseen. Toisin sanoen mittari, jolla laatutekijää ja laadullisen vaatimuksen toteutumista mitataan, on herkkä kyseessä olevan arkkitehtonisen ratkaisun muuttamiselle. Herkkyyspiste on arkkitehtuurin yksityiskohta, johon suunnittelijan täytyy kiinnittää erityistä huomiota, mikäli herkkyyspisteen vaikutuspiirissä olevan laadullisen vaatimuksen saavuttaminen on arvioitu järjestelmän kannalta tärkeäksi.

Kompromissipiste on arkkitehtoninen ratkaisu, joka vaikuttaa useampaan kuin yhteen laatutekijään ja on herkkyyspiste useammalle kuin yhdelle laatutekijälle siten, että vaikutus laatutekijöihin on vastakkainen.

### 3.3.5 Riskit ja ei-riskit

ATAM-menetelmän käsitteistössä riskit ovat arkkitehtonisia suunnittelupäätöksiä, jotka arkkitehtuurin arvioinnin yhteydessä todetaan potentiaalisesti ongelmallisiksi [CKK01]. Vastaavasti ei-riskit ovat arkkitehtonisia suunnittelupäätöksiä, jotka todetaan tarkoituksenmukaisiksi ja toimiviksi niiden laatutekijöiden suhteen, joihin ei-riskit vaikuttavat.

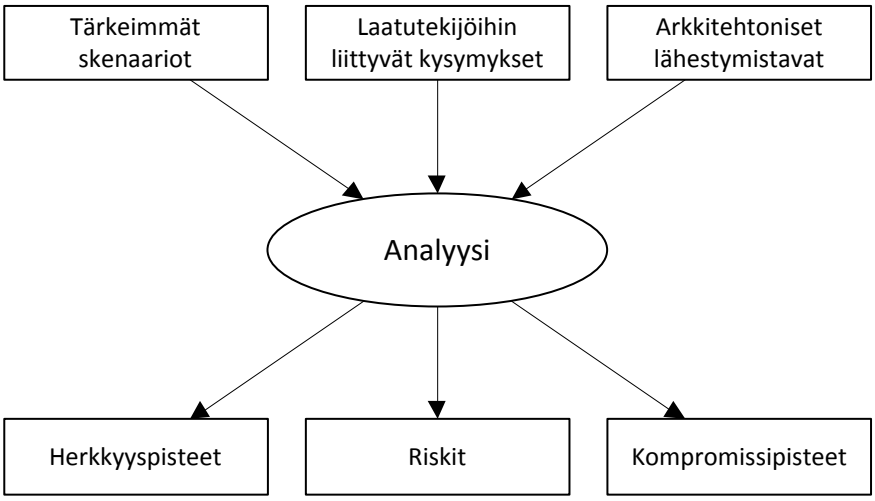
### 3.4 Tärkeimmät tavoitteet

ATAM-menetelmän päätavoite on arvioida arkkitehtonisten päätösten vaikutuksia ohjelmistojärjestelmän laadullisten vaatimusten toteutumiseen [KKC00]. Edellytyksenä arkkitehtuurin arvioinnille ovat laadullisten vaatimusten täsmällinen määrittely sekä arkkitehtuurisuunnitelma, josta selviävät tehdyt arkkitehtuuritason suunnittelupäätökset. Ei ole mitenkään epätavallista, että järjestelmän laadullinen vaatimusmäärittely ja arkkitehtuurisuunnitelma olisi laadittu epämääräisesti tai puutteellisesti. Tästä johtuen ATAM-menetelmän tavoitteisiin kuuluvat myös järjestelmän tärkeimpien laadullisten vaatimusten sekä arkkitehtonisten päätösten selvittäminen ja niitä koskevien kuvausten tarkentaminen [KKC00].

ATAM-menetelmän mukaisen arkkitehtuurianalyysin tärkeimmät tulokset ovat arkkitehtuurin herkkyys- ja kompromissipisteet sekä luettelo arkkitehtuuriin sisältyvistä riskeistä. Lopullinen analyysi perustuu kolmeen prosessin aikana tuotettavaan komponenttiin [CKK01]:

1) tärkeimmät laadullisia vaatimuksia kuvaavat skenaariot
2) tarkennetut laatutekijät sekä niihin liittyvät kysymykset ja informaatio
3) tärkeimpiin laadullisiin vaatimuksiin vaikuttavat arkkitehtoniset ratkaisut

Kuvassa 6 ovat ATAM-menetelmän tuottamat arkkitehtuurianalyysissä käytettävät tiedot, sekä varsinaisen analyysin tärkeimmät lopputulokset.



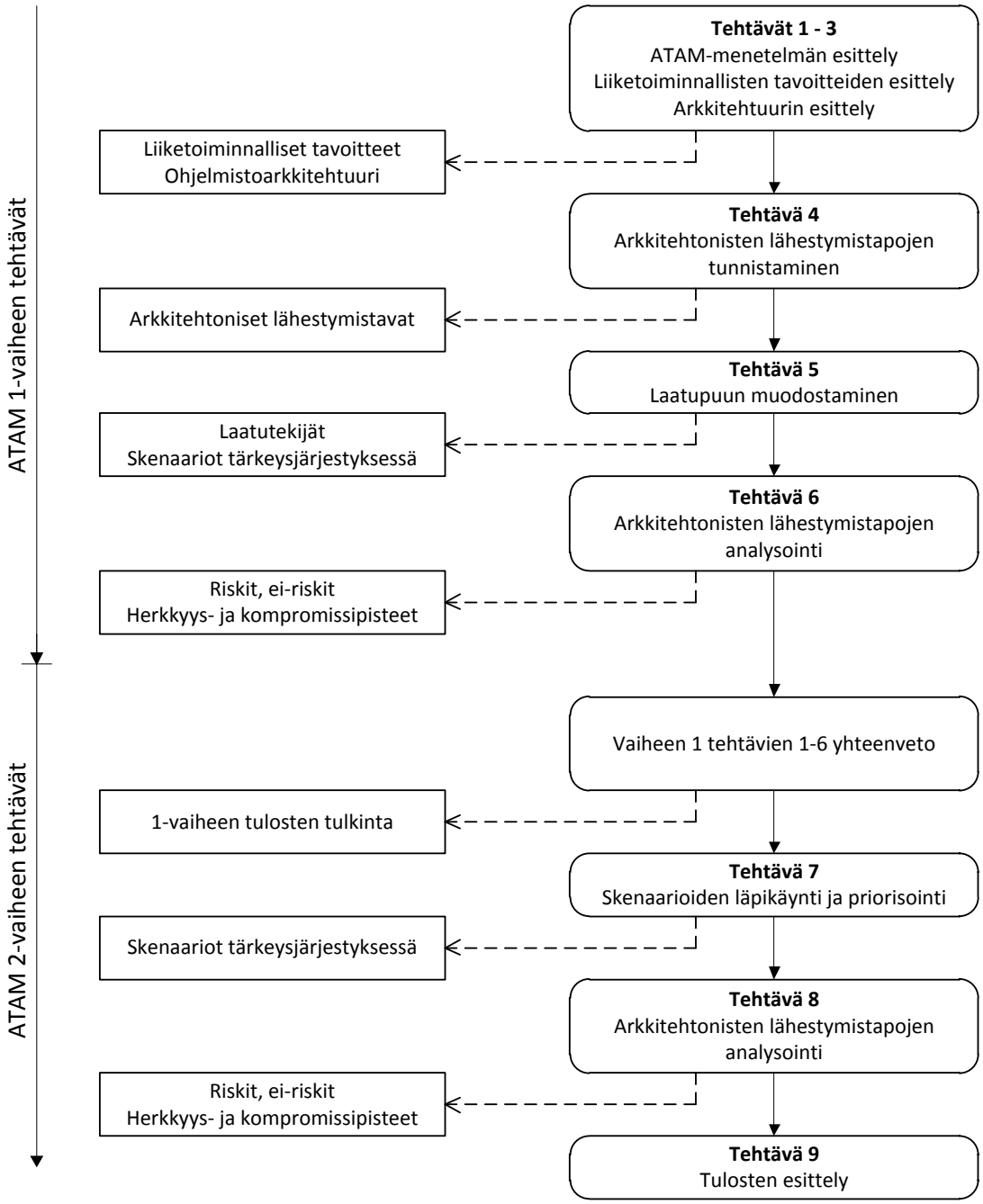

*Kuva 6: ATAM-menetelmän tuottamien tulosten väliset suhteet [KKC00]*

ATAM pyrkii tunnistamaan ja analysoimaan arkkitehtuurin herkkyys- ja kompromissipisteitä, koska niihin sisältyvät päätökset voivat estää vaadittujen laadullisten vaatimusten saavuttamisen [BZJ04]. ATAM myös helpottaa ristiriitaisten laatutavoitteiden tunnistamista ja niiden välillä tehtäviä valintoja [BaG04].

### 3.5 Arvioinnin vaiheet ja tehtävät

ATAM-menetelmän mukaisen arviointiprosessin tehtävät suoritetaan tyypillisesti kahdessa vaiheessa [KKC00]. Ensimmäinen vaihe sisältää kuusi tehtävää, jotka keskittyvät järjestelmän laadullisten vaatimusten ja arkkitehtuurikuvauksen selvittämiseen sekä niitä koskevan dokumentaation laadun varmistamiseen. Ensimmäisen vaiheen yhtenä tavoitteena on varmistaa, että toiseen vaiheeseen siirryttäessä kaikki tarvittava materiaali on saatavilla ja laadultaan riittävää. Toinen vaihe alkaa kertaamalla kaikki ensimmäisen vaiheen tehtävät, joita seuraa neljä arkkitehtuurin analysointiin ja analyysin tulosten valmistamiseen ja esittämiseen keskittyvää tehtävää.

Kuvassa 7 näkyvät menetelmän kaksi peräkkäistä vaihetta ja vaiheiden sisältämät tehtävät. Tehtävät on ryhmitelty sen mukaan mihin arviointimenetelmän tuottamiin lopputuloksiin ne tuottavat informaatiota.




*Kuva 7: ATAM-menetelmän vaiheet ja vaiheisiin kuuluvat tehtävät [BaG04]*

Alla on kuvattu lyhyesti ATAM-menetelmän eri tehtävien sisältö [CKK01].

**1. ATAM-menetelmän esittely**

Arkkitehtuuriarvioinnin vetäjä kuvaa ATAM-menetelmän arviointiin osallistuville henkilöille ja vastaa heidän esittämiinsä kysymyksiin.

**2. Liiketoiminnallisten tavoitteiden esittely**

Sovelluskehitysprojektin projektipäällikkö kertoo, mitkä ovat projektin taustalla olevat liiketoiminnalliset tavoitteet sekä tiedossa olevat, järjestelmän arkkitehtuuriin kohdistuvat vaatimukset.

Tehtävä tuottaa informaatiota

- (alustavaan) priorisoituun listaan eri laatutekijöihin liittyvistä vaatimuksista
- (alustavaan) luetteloon riskeistä ja ei-riskeistä

**3. Arkkitehtuurin esittely**

Kehitettävän järjestelmän ohjelmistoarkkitehti esittelee arkkitehtuurisuunnitelman ja kertoo kuinka arkkitehtuuri vastaa liiketoiminnallisiin tavoitteisiin.

Tehtävä tuottaa informaatiota

- luetteloon arkkitehtonisista lähestymistavoista
- (alustavaan) luetteloon riskeistä ja ei-riskeistä
- (alustavaan) luetteloon herkkyys- ja kompromissipisteistä

**4. Arkkitehtonisten lähestymistapojen tunnistaminen**

Arkkitehti esittelee arkkitehtoniset lähestymistavat, mutta niitä ei analysoida tarkasti.

Tehtävä tuottaa informaatiota

- luetteloon arkkitehtonisista lähestymistavoista
- arkkitehtonisiin lähestymistapoihin ja laatutekijöihin liittyviin kysymyksiin
- (alustavaan) luetteloon riskeistä ja ei-riskeistä
- (alustavaan) luetteloon herkkyys- ja kompromissipisteistä

**5. Laatupuun muodostaminen**

Arviointiryhmä laatii ATAM-menetelmän mukaisen, järjestelmän laadullisia tavoitteita kuvaavan laatupuun. Laatupuun tulee sisältää kaikki määritellyt elementit mukaan lukien skenaariot, jotka järjestetään tärkeysjärjestykseen.

Tehtävä tuottaa informaatiota

- priorisoituun listaan eri laatutekijöihin liittyvistä vaatimuksista

**6. Arkkitehtonisten lähestymistapojen analysointi**

Arviointiryhmä tunnistaa ja analysoi ne arkkitehtoniset lähestymistavat, jotka vaikuttavat tärkeimmiksi arvioituihin skenaarioihin.

Tehtävä tuottaa informaatiota

- (täydentävää informaatiota) luetteloon arkkitehtonisista lähestymistavoista
- arkkitehtonisiin lähestymistapoihin ja laatutekijöihin liittyviin kysymyksiin
- arkkitehtonisten lähestymistapojen ja laatutekijöihin yhdistämiseen
- luetteloon riskeistä ja ei-riskeistä
- luetteloon herkkyys- ja kompromissipisteistä

**7. Skenaarioiden läpikäynti ja priorisointi**

Arviointiryhmä lisää arkkitehtuurin arvioinnin ja lähestymistapojen analysoinnin aikana mahdollisesti laaditut uudet skenaariot laatupuuhun, jonka jälkeen laatupuun sisältämät skenaariot järjestetään uudelleen tärkeysjärjestykseen.

Tehtävä tuottaa informaatiota

- listaan eri laatutekijöihin liittyvistä vaatimuksista

**8. Arkkitehtonisten lähestymistapojen analysointi**

Tehtävä 8 toistaa tehtävän 6 toimenpiteet, jotka kohdistetaan uudelleen järjestetyn laatupuun tärkeimpiin skenaarioihin.

Tehtävä tuottaa informaatiota

- (täydentävää informaatiota) luetteloon arkkitehtonisista lähestymistavoista
- arkkitehtonisiin lähestymistapoihin ja laatutekijöihin liittyviin kysymyksiin
- arkkitehtonisten lähestymistapojen ja laatutekijöihin yhdistämiseen
- luetteloon riskeistä ja ei-riskeistä
- luetteloon herkkyys- ja kompromissipisteistä

**9. Tulosten esittely**

Arviointiryhmä esittelee arvioinnin tulokset projektin ja järjestelmän sidosryhmille.

# 4 Vanhan järjestelmän modernisointi

## 4.1 Ohjelmistojärjestelmän kehittyminen

Mikä tahansa ohjelmistojärjestelmään kohdistuva muutos, joka muuttaa järjestelmää haluttuun suuntaan, on osa ohjelmistojärjestelmän kehittymistä. Ohjelmistojärjestelmän kehittymiseen liittyvät toimenpiteet voidaan jakaa kolmeen luokkaan: järjestelmän ylläpito, järjestelmän modernisointi ja järjestelmän korvaaminen [CWS00, WNS97].

Vanhan järjestelmän ongelmien korjaamiseen käytettyjen menetelmien jako järjestelmän modernisointiin ja järjestelmän korvaamiseen ei ole täysin yksiselitteistä. Modernisointiin kuuluvien menetelmien pitkäkestoinen soveltaminen komponenttitasolla voi johtaa koko järjestelmän korvautumiseen. Menetelmien luokitteluun vaikuttaa olennaisesti vanhan järjestelmän sovelluslogiikan ja komponenttien suunniteltu elinkaari. Modernisoinnissa ainakin osa vanhasta järjestelmästä sisältyy uudistamistoimenpiteiden lopputuloksena syntyvään järjestelmään. Korvaamisessa vanhasta järjestelmästä ei säilytetä mitään.

### 4.1.1 Järjestelmän ylläpito

Järjestelmän ylläpito on jatkuva, iteratiivinen prosessi, jossa järjestelmään toteutetaan pieniä, ennalta suunnittelemattomia muutoksia. Yleensä muutokset liittyvät havaittujen virheiden korjaamiseen tai pieniin järjestelmän toiminnallisuuden lisäyksiin. Ylläpitoon ei pitäisi kuulua suuria, järjestelmän rakenteeseen kohdistuvia muutoksia.

Jatkuva ylläpito on kaikkia järjestelmiä koskeva kehittymisen edellytys. Pelkillä ylläpitotoimilla aikaansaatava järjestelmän kehittyminen on kuitenkin hyvin rajoittunutta. Comella-Dorda et al. kuvaavat seuraavat tekijät, jotka rajoittavat ylläpitotoimilla saavutettavaa järjestelmän kehittymistä [CWS00].

1) Mahdollisuudet kilpailuedun saavuttamiseen uusien teknologioiden avulla ovat erittäin rajoittuneet. Yleensä uusien teknologioiden käyttöönotto vaatii toimenpiteitä, joita ei laajuutensa vuoksi luokitella ylläpitoon kuuluviksi.
2) Vanhan järjestelmän ylläpitokustannukset kasvavat ajan myötä, kun vanhentuneisiin teknologioihin liittyvän osaamisen hankkiminen vaikeutuu ja muuttuu kalliimmaksi.
3) Järjestelmän muuttaminen useilla pienillä ylläpitotoimenpiteillä voi aiheuttaa laajemman yhteisvaikutuksen kuin yksittäisten toimenpiteiden vaikutuksista voisi päätellä. Vanhan järjestelmän muuttaminen vastaamaan muuttuneita liiketoiminnallisia vaatimuksia muuttuu järjestelmän ikääntyessä yhä vaikeammaksi.

### 4.1.2 Järjestelmän modernisointi

Järjestelmän modernisointi tuottaa järjestelmään suurempia muutoksia kuin ylläpitoon kuuluvat toimenpiteet, mutta säilyttää merkittävän osan vanhaa järjestelmää ennallaan. Yleensä modernisointiin kuuluvat toimenpiteet uudistavat järjestelmän rakennetta, laajentavat järjestelmän sisältämää toiminnallisuutta tai vaikuttavat positiivisesti järjestelmän laatutekijöihin. Modernisointi soveltuu uudistamismenetelmäksi järjestelmille, jotka vaativat laajempia muutoksia kuin ylläpitoon voidaan sisällyttää [CWS00].

Järjestelmän modernisointimenetelmät voidaan jakaa kahteen luokkaan sillä perusteella, kuinka tarkkaa vanhan järjestelmän rakenteen ja toiminnan tuntemista ne edellyttävät. Menetelmiä, jotka vaativat järjestelmän sisäisen rakenteen ja toteutuksen tuntemista, kutsutaan *lasilaatikko-modernisoinniksi* (white-box modernization). Menetelmiä, joiden soveltamiseen riittää vanhan järjestelmän ulkoisten rajapintojen toiminnan tunteminen, kutsutaan *musta laatikko -modernisoinniksi* (black-box modernization) [CWS00].

### 4.1.3 Järjestelmän korvaaminen

Järjestelmän korvaaminen tarkoittaa, että järjestelmään kuuluvat sovellukset toteutetaan kokonaisuudessaan uudelleen [BLW99]. Korvaaminen on tarkoituksenmukainen ratkaisu vanhoille järjestelmille, joiden modernisointi ei ole mahdollista tai kustannustehokasta. Korvaamiseen päädytään yleensä sellaisten järjestelmien kohdalla, jotka ovat joko dokumentoimattomia, vanhentuneita tai joita ei enää pystytä laajentamaan [CWS00].

Korvaamiseen sisältyy riskejä, jotka täytyy selvittää ennen menetelmän soveltamista. Comella-Dorda et al. mainitsevat erityisesti kaksi riskiä [CWS00]:

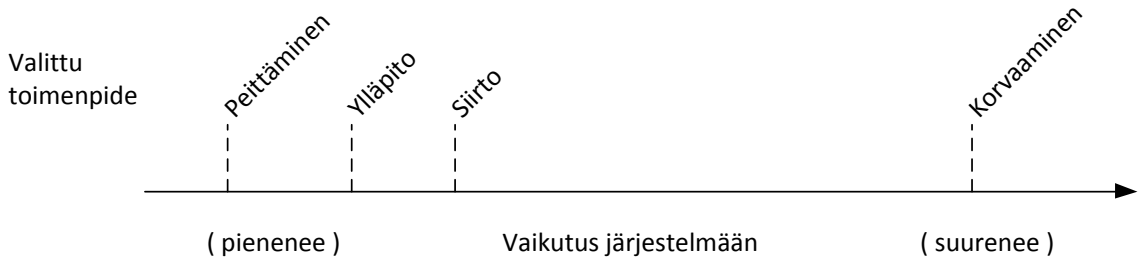
1) Riittämättömät resurssit.
   Korvaaminen on pohjimmiltaan järjestelmän toteuttamista uudelleen tyhjästä aloittaen ja sitoo merkittävästi resursseja. Yrityksen ohjelmistokehittäjät ovat tyypillisesti vanhan järjestelmän ylläpitotehtäviin täysin työllistettyjä, eivätkä he välttämättä tunne korvaavan järjestelmän toteutuksessa käytettäviä teknologioita.
2) Vaaditun toiminnallisuuden varmentaminen.
   Korvaaminen edellyttää uuden, korvaavan järjestelmän laaja-alaista testaamista. Vanhat järjestelmät sisältävät tuotantokäytössä testattua toiminnallisuutta ja niihin on sitoutunut merkittävä määrä liiketoimintaan liittyvää asiantuntemusta. Uusi järjestelmä ei välttämättä ole yhtä toimiva kuin vanha ja sen tarjoama toiminnallisuus voi pysyä jonkin aikaa käyttöönoton jälkeen vanhaa järjestelmää suppeampana.

Korvaamisen käyttöä järjestelmän uudistamismenetelmänä vaikeuttavat myös jatkuvasti muuttuvat teknologiat ja liiketoiminnalliset vaatimukset. Järjestelmän korvaaminen vaatii pitkäkestoisen projektin ja sen lopputuloksena voi olla vanhentuneisiin teknologioihin perustuva järjestelmä, joka ei valmistuessaan täytä voimassa olevia liiketoiminnallisia vaatimuksia [BLW99].

## 4.2 Modernisoinnin toteutustapoja

Useat järjestelmän modernisointimenetelmät voidaan luokitella joko *peittämiseksi* (wrapping) tai *siirroksi* (migration) [BLW99]. Järjestelmän modernisointia kokonaisuutena ei kuitenkaan voida aina määritellä selkeästi tiettyyn luokkaan kuuluvaksi. Vaikka järjestelmätasolla kyse olisi vanhan järjestelmän siirrosta, voidaan komponenttitasolla soveltaa menetelmiä, jotka ovat selkeästi sovelluslogiikan peittämistä tai korvaamista. Komponenttitason muutokset voivat myös olla luonteeltaan sellaisia, että järjestelmän modernisointiprojektin ulkopuolella ne luokiteltaisiin ylläpitotoimenpiteiksi.

Kuvassa 8 on vertailtu vanhan järjestelmän ylläpidon jatkamisesta, kahden eri modernisointimenetelmän soveltamisesta, sekä järjestelmän korvaamisesta aiheutuvien muutosten määrää.




*Kuva 8: Eri toimenpiteiden vaatimat muutokset järjestelmään [BLW99]*

### 4.2.1 Peittäminen

Toteutuksen peittämisellä tarkoitetaan yleensä hyvin matalalle tasolle sovelluslogiikkaan tehtäviä muutoksia. Ohjelmistokomponenteille toteutetaan uusi rajapinta, jolloin komponentit ovat helpommin muiden komponenttien käytettävissä [BLW99]. Peittäminen on usein musta laatikko -modernisointia ja sen avulla pyritään välttämään tarve tuntea tarkasti vanhan järjestelmän sisäinen toteutus [WNS97, ZhY04].

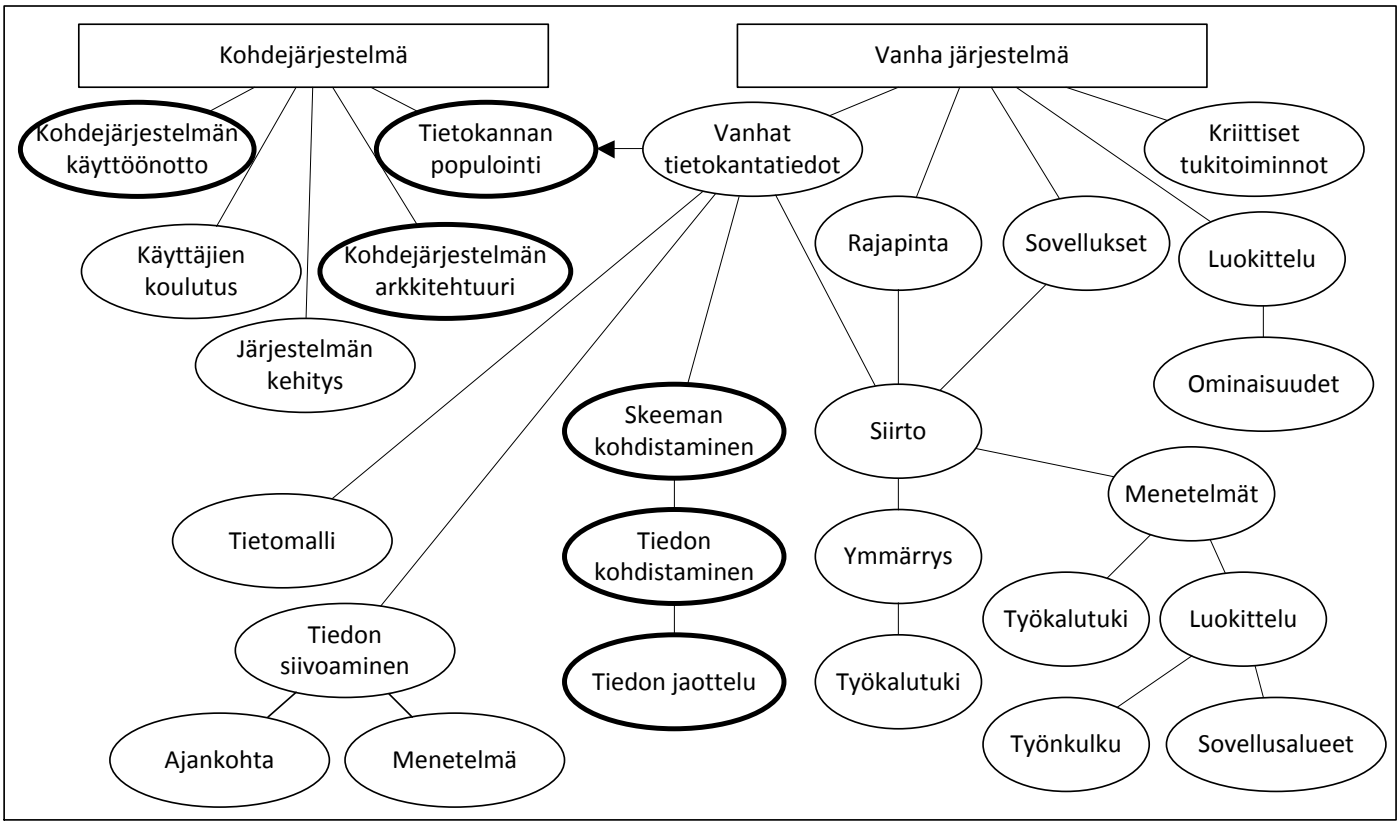
Peittäminen vaatii järjestelmään suhteellisen pieniä muutoksia, mutta usein ratkaisu jää väliaikaiseksi toteutukseksi [BLW99]. Vanhan järjestelmän rakenne voi merkittävästi hankaloittaa tai jopa estää toteutuksen peittämisen. Toteutuksesta voi olla mahdotonta tunnistaa toisistaan erillisiä komponentteja. Myös vanhan järjestelmän epäyhtenäinen toteutustapa voi vaikeuttaa komponenttien peittämistä ja uuden komponenttirajapinnan toteuttamista.

Toisaalta, koska ohjelmistokomponentin määritelmä on varsin väljä, voidaan vanhan toteutuksen peittämiseksi tulkita myös suuremman kokonaisuuden kuin tietyn ohjelmistomoduulin rajapinnan korvaaminen. Vanhan järjestelmän sisäisen toteutuksen tarkka analyysi voidaan jättää tekemättä ja tutkia pelkästään järjestelmän ulointa rajapintaa. Uloimman rajapinnan ympärille toteutetaan ohjelmistokerros, joka peittää vanhan järjestelmän palvelut kokonaisuudessaan ja tarjoaa ne edelleen esimerkiksi web service -rajapinnan kautta. Lähestymistavan huono puoli on järjestelmän muuttuminen monimutkaisemmaksi. Ratkaisun soveltaminen voi pitkällä aikavälillä monimutkaistaa vanhan järjestelmän rakennetta ja lisätä ylläpitoon kuluvaa aikaa [ZhY04].

### 4.2.2 Siirto

Vanhan järjestelmän siirrolla tarkoitetaan käytössä olevan ohjelmistojärjestelmän siirtämistä joustavampaan suoritusympäristöön säilyttäen samalla järjestelmän sisältämä tieto ja toiminnallisuus. Siirron tulisi aiheuttaa niin vähän häiriötä järjestelmän käytölle kuin mahdollista. Vaikka siirto on huomattavasti monimutkaisempi tehtävä kuin järjestelmän peittäminen, tuottaa se onnistuessaan pitkäkestoisemman ja paremman ratkaisun [BLW99]. Vanhan järjestelmän toteutus joudutaan analysoimaan siirron yhteydessä perusteellisemmin kuin järjestelmää peitettäessä, mikä voidaan nähdä positiivisena tekijänä. Siirron yhteydessä voidaan tehdä laatua parantavia muutoksia järjestelmän sisäiseen toteutukseen.

Vanhan järjestelmän siirtoon sisältyy lukuisia käytännön ongelmia. Kuvassa 9 on jaettu todennäköisimpiä ongelma-alueita kahteen luokkaan sen mukaan, liittyvätkö ongelmat vanhaan järjestelmään vai siirron kohteena olevaan järjestelmään.




*Kuva 9: Vanhan järjestelmän siirtoon liittyvien ongelmien luokittelua [BLW99]*

Tutkielman esimerkkitapauksena käytettävässä projektissa erityisen riskialttiiksi todetut ongelma-alueet ovat kuvassa 9 vahvennettuina. Tapaustutkimuksessa arvioidaan

nimenomaan kohdejärjestelmän arkkitehtuuria ja arkkitehtuurin soveltuvuutta vanhan järjestelmän modernisointiin. Arvioinnissa kuitenkin oletetaan, että arkkitehtuuria koskevilla suunnittelupäätöksillä voidaan vaikuttaa muidenkin ongelma-alueiden ongelmiin.

Työllistävimmät ongelmat vanhan järjestelmän siirrossa liittyvät usein tietokannan rakenteen ja tietojen siirtoon järjestelmien välillä sekä kohdejärjestelmän testaamiseen [BLW99]. Kohdejärjestelmän testaamisen helpottamiseksi on olemassa useita erilaisia strategioita. Järjestelmän oikeellinen toiminta on helpompi varmentaa, jos toiminnallisia vaatimuksia ei siirron yhteydessä laajenneta.

Järjestelmän siirron viimeinen vaihe on kohdejärjestelmän käyttöönotto. Bisbal et. al [BLW99] kuvaa kolme erilaista käyttöönoton strategiaa.

1) Välitön järjestelmästä toiseen siirtyminen (cut-and-run).
   Vanha järjestelmä kytketään pois käytöstä ja samalla kaikki toiminnallisuuden tuottaminen siirtyy täysin kohdejärjestelmän vastuulle.
2) Vaiheittainen siirto (phased interoperability).
   Siirto toteutetaan vaiheittain, jolloin jokaisen vaiheen aikana korvataan yksi tai useampia vanhan järjestelmän komponentteja kohdejärjestelmän vastaavilla osilla. Siirrettävät komponentit voivat olla esimerkiksi erillisiä sovelluksia tai tietokokonaisuuksia.
3) Rinnakkainen suoritus (parallel operations).
   Vanha järjestelmä ja kohdejärjestelmä ovat yhtä aikaa suorituksessa ja molemmat järjestelmät tarjoavat samat toiminnot. Rinnakkaisen suorituksen aikana kohdejärjestelmää testataan jatkuvasti. Kun kohdejärjestelmän oikeellinen toiminta on varmennettu, kytketään vanha järjestelmä pois käytöstä.

Välitön järjestelmästä toiseen siirtyminen on useimmiten epärealistinen vaihtoehto siihen sisältyvän suuren riskin takia [BLW99]. Kohdejärjestelmää on haastavaa testata kattavasti ennen siirron toteuttamista.

Rinnakkainen suoritus pienentää testaukseen liittyvää riskiä, mutta toisaalta strategia vaatii, että vanha järjestelmä ja kohdejärjestelmä ovat toiminnoiltaan lähes identtisiä. Jotta paluu vanhaan järjestelmään olisi vikatilanteessa mahdollista, on järjestelmien tuottaman tiedon oltava yhteensopivaa tai muunnettavissa yhteensopivaksi.

Myös vaiheittainen siirto pienentää kohdejärjestelmän testaukseen ja sen mahdollisiin virheisiin liittyvää riskiä. Vaatimukset vanhan järjestelmän ja kohdejärjestelmän

toteutusten samankaltaisuudesta ovat lievemmät kuin rinnakkaisessa suorituksessa. Onnistuakseen strategia vaatii, että vanha järjestelmä voidaan pilkkoa erillisiin sovelluksiin tai toiminnallisiin osakokonaisuuksiin, jotka ovat siirrettävissä toisistaan riippumattomasti. Useimpien vanhojen järjestelmien monoliittinen ja jäsentelemätön toteutus tekee vaiheittaisesta siirrosta vaikeaa tai jopa mahdotonta [BLW99]. Eri järjestelmissä suoritettavat sovellukset voivat myös olla riippuvaisia samoista resursseista, mikä osaltaan vaikeuttaa strategian soveltamista.

### 4.3 Modernisointiin johtavat tekijät

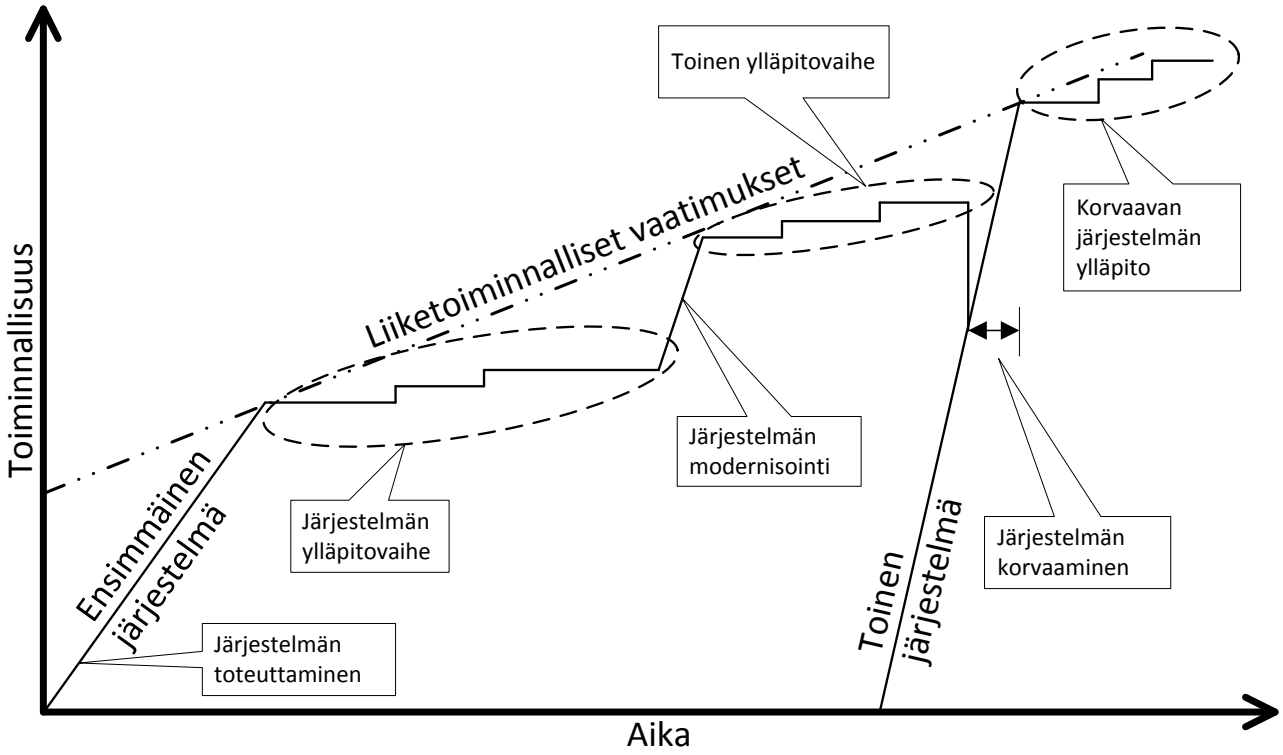
Vanha järjestelmä ei välttämättä vaadi niin paljon ylläpito- tai muutostöitä, että sen osittainenkaan korvaaminen olisi perusteltua. Mitä tärkeämpi rooli vanhalla järjestelmällä on yrityksen liiketoiminnassa, sitä todennäköisemmin järjestelmää koskevat laadulliset ja toiminnalliset vaatimukset tulevat muuttumaan liiketoiminnallisten tavoitteiden muuttumisen seurauksena. Vanhan järjestelmän muuttamiseen ja ylläpitoon liittyy useita haasteita. Bisbal et al. mainitsevat erityisesti neljä ongelmaa [BLW99]:

1) Yleensä vanhoja järjestelmiä suoritetaan vanhentuneella laitteistolla, jonka ylläpito on hidasta ja kallista.
2) Dokumentaation puuttuminen ja/tai järjestelmän yksityiskohtien puutteellinen tunteminen voivat vaikeuttaa ohjelmiston ylläpitoa ja tehdä virheiden jäljittämisestä kallista sekä aikaa vievää.
3) Selkeiden rajapintojen puuttuminen heikentää vanhan järjestelmän integroitavuutta muihin järjestelmiin.
4) Vanhaa järjestelmää on vaikea, ellei mahdoton laajentaa.

Yrityksen kehittyessä ja muuttuessa myös liiketoiminnan kannalta kriittisiltä vanhoilta järjestelmiltä edellytetään vastaavaa kehitystä ja muutosta. Järjestelmien muuttuessa tulisi niiden laadun pysyä vähintään tyydyttävällä tasolla, jotta järjestelmien ylläpito olisi kustannustehokasta. Vanhojen järjestelmien laatu on erittäin usein huonontunut pitkään jatkuneen kehityksen aikana ja huonosta laadusta johtuen järjestelmien ylläpito voi olla erittäin kallista [BCM03]. Yrityksen liiketoiminnan kehittyminen voi siis edellyttää liiketoimintaa tukevan vanhan järjestelmän modernisointia, vaikka järjestelmä täyttäisikin sille asetetut toiminnalliset vaatimukset. Modernisoinnin perusteeksi riittää vanhan järjestelmän laadullisten tavoitteiden muuttuminen, joka voi johtua pelkästään muuttuneista liiketoiminnallisista vaatimuksista.

Järjestelmän modernisointiprojekti voi olla hyvin laaja ja pitkäkestoinen, ja siitä voi aiheutua yritykselle merkittäviä kustannuksia. Kustannustehokkuussyistä modernisoinnin tulisikin laadun parantamisen ohella mahdollistaa vanhan järjestelmän toiminnallisuuden laajentaminen sekä uusien teknologioiden käyttöönottaminen [BCM03].

Ohjelmistojärjestelmän muuttuviin liiketoiminnallisiin vaatimuksiin voidaan vastata pelkillä ylläpitotoimenpiteillä vain rajatun ajan. Kun järjestelmän sisältämä toiminnallisuus on jäänyt riittävän paljon jälkeen vaatimuksista, joudutaan vanha järjestelmä modernisoimaan. Modernisointi laajentaa järjestelmän toiminnallisuutta niin paljon, että muuttuvat liiketoiminnalliset vaatimukset voidaan taas jonkin aikaa täyttää pelkällä ylläpidolla [CWS00]. Lopulta vanhan järjestelmän kehittyminen muuttuvien liiketoiminnallisten vaatimusten mukaisesti ei ole enää mahdollista ja järjestelmä joudutaan korvaamaan. Kuvassa 10 on esitetty ohjelmistojärjestelmän toiminnallisuuden laajentamiseksi käytettäviä toimenpiteitä järjestelmän elinkaaren eri vaiheissa.




*Kuva 10: Ohjelmistojärjestelmän elinkaari [CWS00]*

# 5 Esimerkkitapaus

## 5.1 Modernisointiprojekti

Tutkielmassa käsiteltävä esimerkkitapaus on erään ohjelmistoalan yrityksen vanhan ohjelmistojärjestelmän modernisointiprojekti. Projektin tavoitteena oli paitsi järjestelmän laadun parantaminen, myös ohjelmistokehityksen nopeuttaminen ja kehitystyöhön sisältyvän laadunvalvonnan parantaminen uudistamalla käytettäviä teknologioita, työkaluja ja prosesseja. Projektin liiketoiminnalliset tavoitteet on kuvattu tarkemmin luvussa 5.5.

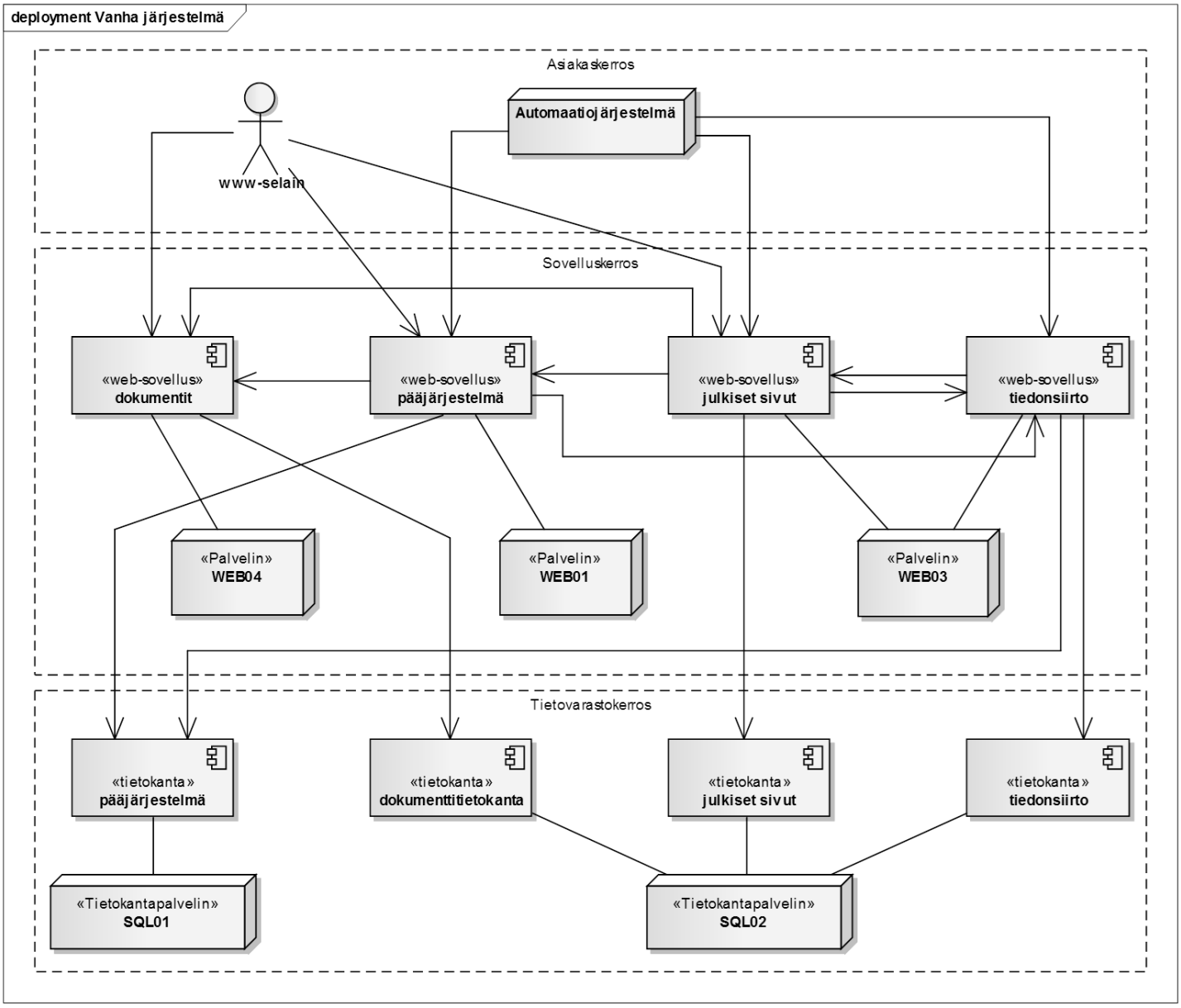
Esimerkkitapaus vastaa kuvan 10 mukaisessa ohjelmistojärjestelmän elinkaarimallissa korvaavan järjestelmän toteuttamista, joka päättää vanhan järjestelmän ylläpitovaiheen. Vanhan järjestelmän laajuuden ja kriittisen liiketoiminnallisen merkityksen vuoksi modernisointiprojektin arvioitiin olevan hyvin pitkäkestoinen. Vanhan ja korvaavan järjestelmän tuli yhdessä muodostaa tuotantokelpoinen järjestelmä koko projektin ajan, eikä vanhan järjestelmän ylläpitoa saanut keskeyttää korvaavan järjestelmän suunnittelu- ja kehitystyön ajaksi.

## 5.2 Vanha järjestelmä

Vanha järjestelmä on kiinteistöalan eri toimijoiden liiketoimintaprosessien ohjaukseen ja liiketoimintatiedon hallintaan tarkoitettu ohjelmistojärjestelmä. Järjestelmän sisältämät sovellukset muodostavat yhtenäisen, internetin kautta tarjottavan palvelun. Asiakasyritysten henkilökunta käyttää järjestelmää yleisimmin henkilökohtaisilla tietokoneilla ja verkkoselaimilla, mutta järjestelmän tietyt osat on suunniteltu käytettäväksi myös matkapuhelimilla ja muilla mobiilipäätelaitteilla. Järjestelmä on yhteydessä eri automaatiojärjestelmiin, jotka välittävät järjestelmään esimerkiksi kiinteistöjen energiankulutukseen, vikailmoituksiin ja automaattihälytyksiin liittyvää tietoa.

### 5.2.1 Vanhan järjestelmän laitteistotason arkkitehtuuri

Vanhan järjestelmän laitteistotason toteutus noudattaa 3-kerrosarkkitehtuuria (3-tier architecture). Järjestelmän jako kerroksiin, tärkeimmät laitteistotason komponentit ja komponenttien väliset yhteydet on esitetty kuvassa 11.




*Kuva 11: Vanhan järjestelmän rakenne laitteisto- ja osajärjestelmätasolla*

Asiakaskerros muodostuu käyttäjien tietokoneiden verkkoselaimista, muista päätesovelluksista ja -laitteista, sekä automaatiojärjestelmistä. Kaikki asiakaskerroksen komponentit käyttävät sovelluskerroksen palveluja internetin kautta. Tiedonsiirto asiakaskerroksen ja sovelluskerroksen välillä tapahtuu suurelta osin HTML- ja SOAP-standardien mukaisessa muodossa.

Sovelluskerros koostuu kolmesta fyysisestä palvelimesta, joilla suoritetaan neljää erillistä web-sovellusta. Sovellusten välillä on erilaisia keskinäisiä riippuvuuksia liittyen

esimerkiksi käyttöoikeuksien valvontaan ja tietojen siirtämiseen. Web-sovellukset käyttävät tietovarastonaan neljää relaatiotietokantaa, joita suoritetaan kahdella fyysisellä tietokantapalvelimella.

Modernisoinnin kohteena oli pelkästään pääjärjestelmäksi kutsuttu web-sovellus sekä sen tietovarastona toimiva tietokanta. Pääjärjestelmän modernisointi ei saanut vaikuttaa muiden sovellusten toimintaan eikä toteutukseen. Pääjärjestelmä on toteutettu Microsoft ASP Classic -teknologialla ja Microsoft SQL Server -tietokantaproseduureilla.

## 5.3 Modernisoinnin toteutustapa

Esimerkkitapauksen modernisoinnin toteutustavan luokittelu riippuu siitä, millä tarkkuustasolla modernisointia tarkastellaan. Järjestelmätasolla on perusteltua käyttää termiä vanhan järjestelmän **korvaaminen**, koska modernisointiprojektin lopullisena tavoitteena oli luopua vanhan järjestelmän käytöstä kokonaan. Vanha järjestelmä korvattaisiin vaiheittain jakamalla se toisistaan riippumattomiin loogisiin sovelluksiin, jotka modernisoitaisiin **siirtämällä** ne osaksi uutta, korvaavaa järjestelmää. Vanhan ja korvaavan järjestelmän yhteentoimivuuden varmistamiseksi eräitä vanhan järjestelmän sovelluksia ja sovellusten komponentteja **peitettäisiin**, jotta niiden sisältämä logiikka olisi kutsuttavissa myös korvaavasta järjestelmästä. Modernisointiprojektin aikana korvaavaan järjestelmään päätettiin määritellä ja toteuttaa uutta, vanhasta järjestelmästä puuttuvaa toiminnallisuutta.

Esimerkkitapauksen pääasiallisena järjestelmän modernisoinnin toteutustapana voidaan pitää sovellusten siirtoa. Bisbal et. al [BLW99] mainitsevat kolme siirtoon liittyvää kriittistä tehtäväkokonaisuutta. Seuraavissa luvuissa kerrotaan lyhyesti, kuinka nämä tehtäväkokonaisuudet suunniteltiin esimerkkitapauksessa tehtäväksi.

### 5.3.1 Tietojen siirto vanhasta järjestelmästä korvaavaan järjestelmään

Vanhaan järjestelmään kuuluvan pääjärjestelmän tietokanta ja sen sisältämät tiedot olivat liiketoiminnalliselta merkitykseltään järjestelmän tärkein siirrettävä kokonaisuus. Siirron yhteydessä pyrittiin ratkaisemaan tietokannan rakenteessa havaittuja ongelmia, eikä korvaavan järjestelmän tietokantaa toteutettu vanhan kannan rakennetta suoraan noudattaen. Uuden tietokannan toteutus oli osa siirrettävien sovellusten kehitystyötä ja tietokannan rakenne päätettiin suunnitella vaiheittain, siirrettävien sovellusten vaatimuksia noudattaen.

Sovellusten tarvitseman tiedon siirtämiseen päätettiin käyttää useita eri strategioita. Harvoin muuttuvat tiedot siirrettäisiin ajastetusti eräajoina. Kriittiset, usein muuttuvat tiedot synkronoitaisiin vanhasta tietokannasta uuteen tietokantaan reaaliaikaisesti käyttäjän suorittamien toimenpiteiden yhteydessä. Reaaliaikaisesti synkronoitavan tiedon määrää rajoitettiin käyttöoikeuksien ja liiketoiminnallisten näkyvyyssääntöjen perusteella. Olennainen osa tiedonsiirtostrategiaa oli vanhan järjestelmän tietokannan tietojen ja relaatioiden perusteellinen analyysi. Analyysin perusteella siirrettävä tieto pyrittiin jakamaan toisistaan riippumattomiin kokonaisuuksiin. Siirron yhteydessä oli myös tarkoitus parantaa järjestelmän sisältämän tiedon laatua rajaamalla siirrettävää tietoa järjestelmän aiemman käytön aikana havaittujen epäkohtien perusteella.

### 5.3.2 Korvaavan järjestelmän testaaminen

Ohjelmistojärjestelmän modernisoinnissa sovelluslogiikan siirtoa suorittava työntekijä voi käyttää jopa 80 prosenttia työajastaan korvaavan järjestelmän testaamiseen [BLW99]. Testauksen vaatiman työmäärän vähentämiseksi esimerkkitapauksen modernisointiprojektin aikana suunniteltiin ja otettiin käyttöön automaattista, jatkuvaa integraatiotestausta tukeva kehitysympäristö. Korvaavan järjestelmän toteutukseen käytettävät teknologiat ja työkalut valittiin siten, että yksikkö- ja skenaariotestien kattavuus saatiin mahdollisimman suureksi.

Bisbal et. al [BLW99] mukaan siirron kohteena oleva järjestelmä tulisi toteuttaa siten, että sen tuottama tulos (output) olisi täysin yhdenmukainen vanhan järjestelmän samalla syötteellä tuottaman tuloksen kanssa. Tällöin korvaavan järjestelmän toiminnan oikeellisuus voitaisiin varmentaa vertaamalla järjestelmien toisiaan vastaavilla syötteillä tuottamia tuloksia. Esimerkkitapauksessa tätä vaatimusta ei noudatettu, koska siirrettävien sovellusten toiminnallisuudessa havaittuja puutteita ja virheitä päätettiin korjata siirron yhteydessä. Lisäksi sovellusten siirto päätettiin aikatauluttaa siten, että käyttäjille ei tarjottaisi samoja palveluja yhtä aikaa sekä vanhasta että korvaavasta järjestelmästä.

### 5.3.3 Korvaavan järjestelmän käyttöönotto

Esimerkkitapauksessa korvaavan järjestelmän käyttöönotossa päätettiin noudattaa luvussa 4.2.2 kuvattua vaiheittaista siirtoa. Menetelmää pidetään monimutkaisuutta lisäävänä ja siksi riskialttiina [BLW99]. Esimerkkitapauksessa vaiheittaisen siirron ongelmien arvioitiin liittyvän erityisesti niihin tietokantatietoihin, joita korvaavaan järjestelmään

siirretyt sovellukset ja edelleen vanhassa järjestelmässä suoritettavat sovellukset käyttäisivät yhtä aikaa.

## 5.4 Korvaavan järjestelmän toiminnalliset vaatimukset

Korvaavan järjestelmän toiminnalliset vaatimukset perustuvat pääosin vanhan järjestelmän sovellusten vaatimusmäärittelyyn. Esimerkkitapauksessa pidettiin todennäköisenä, että korvaavaan järjestelmään toteutettaisiin modernisointiprojektin aikana uusia sovelluksia, vaikka vanhan järjestelmän kaikkia sovelluksia ei olisi vielä siirretty. Myös vanhan järjestelmän toiminnallisten vaatimusten muuttumista projektin aikana pidettiin todennäköisenä, mikä osaltaan vaikutti päätökseen, että korvaavan järjestelmän toiminnalliset vaatimukset määriteltäisiin vaiheittain.

Tutkielmassa korvaavan järjestelmän arkkitehtuurin soveltuvuutta arvioidaan laadullisten vaatimusten toteutumisen näkökulmasta. Laatuvaatimuksia kuvaavissa skenaarioissa käytetään lähinnä vanhan järjestelmän sisältämien sovellusten toiminnallisia vaatimuksia. Skenaarioiden tarkoituksena on sitoa laadulliset vaatimukset realistiseen kontekstiin, eivätkä niissä käytetyt toiminnallisuuden kuvaukset aina vastaa korvaavan järjestelmän lopullisia toiminnallisia vaatimuksia.

## 5.5 Modernisointiprojektin liiketoiminnalliset tavoitteet

Ohjelmistoarkkitehtuurin arviointi ATAM-menetelmällä edellyttää arkkitehtuuria käyttävän ohjelmistojärjestelmän liiketoiminnallisten tavoitteiden kartoittamista [BaG04]. Liiketoiminnalliset tavoitteet täytyy kartoittaa riippumatta siitä, käytetäänkö arvioitavaa arkkitehtuuria täysin uuden järjestelmän toteuttamiseen vai vanhan järjestelmän modernisoinnin tuloksena syntyvän järjestelmän toteuttamiseen.

Esimerkkitapauksen modernisointiprojektin liiketoiminnalliset tavoitteet määriteltiin yrityksen johtoryhmässä ennen projektin alkua. Tavoitteet kuvattiin hierarkkisesti kahdessa tasossa: *korkean tason liiketoiminnallisina tavoitteina* ja *tarkennettuina liiketoiminnallisina tavoitteina*. Samalla määriteltiin tavoitteiden välinen tärkeysjärjestys. Liiketoiminnallisten tavoitteiden luokittelun ja analysoinnin avulla pyrittiin löytämään ne laatutekijät, joiden toteutumisella olisi merkittävä vaikutus tärkeimmiksi arvioitujen liiketoiminnallisten tavoitteiden toteutumiseen. Oletettu laatutekijöiden ja liiketoiminnallisten tavoitteiden välinen syy-seuraussuhde on esitetty kuvassa 12. Eri luokkiin sijoitetut liiketoiminnalliset tavoitteet on kuvattu seuraavissa luvuissa.


*Kuva 12: Laatutekijöiden suhde liiketoiminnallisten tavoitteiden saavuttamiseen*

### 5.5.1 Korkean tason liiketoiminnalliset tavoitteet

Modernisointiprojektin alkuperäiset, korkean tason liiketoiminnalliset tavoitteet on kuvattu alla. Tavoitteet on lueteltu tärkeysjärjestyksessä, yrityksessä suoritetun arvioinnin mukaisesti.

1) **Ylläpitoon kuluvien resurssien vähentäminen**
   Esimerkkitapauksen yrityksessä vallitsevan näkemyksen mukaan vanhan järjestelmän kehittämiseen ja ylläpitoon käytettävästä työajasta kuluu liian suuri osuus ylläpitotehtäviin. Tarkkoja mittauksia työajan jakautumisesta eri työtehtävien väillä ei ole tehty, joten ylläpitoon kuluvien resurssien vähentämiselle ei aseteta modernisointiprojektissa tarkkaa, henkilötyöajassa mitattavaa tavoitetta.
   Ylläpitoon kuluvien resurssien vähentäminen todetaan kuitenkin modernisointi-projektin tärkeimmäksi liiketoiminnalliseksi tavoitteeksi. Tavoitteena ei ole vähentää yrityksen henkilöstöä, vaan vähentää ylläpitotehtäviin kuluvaa työaikaa.
2) **Sovellusten julkaisun nopeuttaminen (time-to-market)**
   Vanha järjestelmä koostuu useista erillisistä sovelluksista. Toiminnallisten vaatimusten muuttuessa järjestelmää laajennetaan toteuttamalla ja lisäämällä siihen uusia sovelluksia. Yrityksen arvion mukaan uusien vaatimusten määrittelyn, ja vaatimuksia vastaavien, uusien sovellusten julkaisun välinen aika on liian pitkä. Modernisointiprojektin tavoitteena on lyhentää sovellusten suunnitteluun, toteuttamiseen ja julkaisuun kuluvaa aikaa.
3) **Tuotannon tehostaminen**
   Yrityksen arvion mukaan useat vanhan järjestelmän toteutukseen ja kehityksessä käytettäviin teknologioihin liittyvät ominaisuudet heikentävät ohjelmistotuotannon tehokkuutta. Tuotannon tehokkuutta ei yrityksessä ole mitattu tarkasti, joten tuotannon tehostamiselle ei aseteta modernisointiprojektissa tarkasti mitattavia tavoitteita. Modernisointiprojektin aikana tehtävissä arkkitehtonisissa ratkaisuissa pyritään huomioimaan ratkaisujen vaikutus tuotannon tehokkuuteen.
4) **Asiakastyytyväisyyden parantaminen**
   Modernisointiprojektin tuloksena syntyvä korvaava järjestelmä pyritään toteuttamaan siten, että tiedossa olevat, asiakastyytyväisyyttä vähentävät vanhan järjestelmän ongelmat tullaan ratkaisemaan.

5) **Pääsy uusille sovellusalueille**
   Tietyt vanhan järjestelmän toteutukseen ja kehityksessä käytettäviin teknologioihin liittyvät ominaisuudet estävät järjestelmän käyttämisen liiketoiminnallisesti tärkeiksi arvioiduilla sovellusalueilla. Modernisointiprojektin tuloksena syntyvän korvaavan järjestelmän toteutuksesta pyritään eliminoimaan tällaiset rajoitukset.

### 5.5.2 Tarkennetut liiketoiminnalliset tavoitteet

Korkean tason liiketoiminnalliset tavoitteet todettiin yrityksessä liian abstrakteiksi ja tavoitteita pyrittiin konkretisoimaan laatimalla joukko tarkennettuja liiketoiminnallisia tavoitteita. Tarkennetut liiketoiminnalliset tavoitteet on kuvattu alla. Tavoitteet on lueteltu yrityksessä tehdyn arvioinnin mukaisessa tärkeysjärjestyksessä.

1) **Helpottaa laadunvalvontaa, sekä nopeuttaa ja aikaistaa järjestelmän virheiden havaitsemista**
   Vanhan järjestelmän rakenne ja käytössä oleva ohjelmistokehitysprosessi eivät mahdollista riittävän tehokasta laadunvalvontaa. Modernisointiprojektin aikana tehtävillä ratkaisuilla pyritään korjaamaan vanhan järjestelmän sisältämiä, laadunvalvontaa ja virheiden havaitsemista vaikeuttavia tekijöitä.
2) **Helpottaa uusien teknologioiden ja innovaatioiden käyttöönottoa järjestelmän kehityksessä**
   Yrityksessä koetaan ongelmaksi vanhan järjestelmän yhteensopimattomuus useiden, vastaavissa järjestelmissä yleisesti käytössä olevien teknologioiden kanssa. Ohjelmistoalalla hyväksi havaittujen, uusien teknologioiden käyttöönotosta pyritään modernisointiprojektin aikana tekemään aiempaa helpompaa.
3) **Helpottaa valmiiden ratkaisujen hyödyntämistä järjestelmän kehityksessä**
   Kolmansien osapuolien toteuttamien komponenttien liittäminen vanhaan järjestelmään on koettu liian vaikeaksi ja järjestelmän muutettavuutta pyritään tältä osin parantamaan modernisointiprojektin aikana.
4) **Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen järjestelmän ominaisuuksiin ja toimintaan sekä yrityksen liiketoiminta-alueeseen**
   Yrityksen arvion mukaan korvaavan järjestelmän rakenteessa on mahdollista tehdä ratkaisuja, jotka helpottavat järjestelmän toiminnan, ominaisuuksien ja eri sovellusten liiketoiminnallisen merkityksen ymmärtämistä.

5) **Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen käytettäviin teknologioihin ja työkaluihin**
   Vanhan järjestelmän toteutuksessa käytetyt teknologiat ja työkalut ovat yrityksen arvion mukaan perusteettoman vaikeasti omaksuttavissa ja opittavissa. Korvaavan järjestelmän toteutuksessa nämä ongelmat pyritään korjaamaan.
6) **Työskentelyprosessien uudistaminen**
   Yrityksessä arvioitiin tärkeäksi, että modernisointiprojektin aikana tuotettava järjestelmän rakenne tukee yrityksen työskentelyprosessien muokkaamista aiempaa enemmän ryhmätyötyyppiseksi.
7) **Helpottaa uusien henkilöresurssien hankkimista rekrytoimalla**
   Vanhan järjestelmän toteutukseen käytettyjen teknologioiden osaajista on yrityksen arvion mukaan pulaa, mikä on osaltaan vaikeuttanut uusien kehittäjien rekrytointia. Nämä ongelmat pyritään estämään korvaavan järjestelmän teknologiavalinnoilla.
8) **Helpottaa ulkopuolisten henkilöresurssien käyttämistä kehitystyössä sekä mahdollistaa järjestelmän kehitysprojektien ulkoistaminen**
   Vanhan järjestelmän rakenne ja toteutusteknologia hankaloittavat itsenäisten ohjelmistokomponenttien liittämistä järjestelmään, mikä puolestaan vaikeuttaa kehitystyön ulkoistamista. Modernisointiprojektin yhtenä tavoitteena on helpottaa kehitystyön ulkoistamista.

Tarkennetut liiketoiminnalliset tavoitteet johdettiin korkean tason tavoitteista siten, että jokainen tarkennettu tavoite tukee toteutuessaan yhden tai useamman korkean tason tavoitteen toteutumista. Vaatimusmäärittelyn mukaiset korkean tason ja tarkennettujen liiketoiminnallisten tavoitteiden väliset suhteet on kuvattu tutkielman liitteessä 1.

## 5.6 Ohjelmistoarkkitehtuurin soveltuvuuden arviointi

Esimerkkitapauksen modernisointiprojektissa ohjelmistoarkkitehtuurin soveltuvuutta vanhan järjestelmän vaiheittain korvaavan järjestelmän toteuttamiseen arvioitiin ennen varsinaisen toteutustyön aloittamista. Arvioinnilla pyrittiin selvittämään kuinka hyvin suunniteltu ohjelmistoarkkitehtuuri tukee projektille määriteltyjen liiketoiminnallisten tavoitteiden, sekä niistä järjestelmälle johdettujen laadullisten tavoitteiden saavuttamista. Arvioinnissa päätettiin käyttää tutkielman luvussa 3 kuvattua ATAM-menetelmää.

# 6 Korvaavan järjestelmän laadulliset vaatimukset

## 6.1 Laadulliset vaatimukset ja laatutekijät

Ohjelmistoarkkitehtuuria arvioimalla pyritään selvittämään minkälaiset mahdollisuudet arkkitehtuuri antaa sellaisen järjestelmän toteuttamiselle, joka pystyy täyttämään sille asetetut laadulliset vaatimukset [BZJ04]. ATAM-menetelmä keskittyy laatutekijöiden avulla kuvattujen laadullisten vaatimusten toteutumisen arviointiin [KKC00]. Laatutekijöiden toteutuminen järjestelmässä määrittää järjestelmän kokonaislaadun [IEE98]. Valitut laatutekijät määritellään arviointiprosessin yhteydessä, jolloin varmistetaan, että kaikki arviointiin osallistuvat henkilöt käsittävät laatutekijät samalla tavalla. Esimerkkitapauksessa käytettyjen laatutekijöiden määrittelyt ovat luvussa 2.2.

Laatutekijöiden merkitys korostuu eri tavoin eri ohjelmistojärjestelmissä, riippuen esimerkiksi järjestelmän käyttötarkoituksesta. Esimerkkitapauksessa tärkeimpien laatutekijöiden valinta perustui järjestelmän liiketoiminnallisten tavoitteiden analysointiin. Analysoinnissa oletettiin, että järjestelmän laadullisiin ominaisuuksiin vaikuttamalla voidaan vaikuttaa myös liiketoiminnallisten tavoitteiden saavuttamisen todennäköisyyteen (kuva 12). Analysoinnin tuloksena saadut laatutekijöiden ja tarkennettujen liiketoiminnallisten vaatimusten väliset vaikutussuhteet on kuvattu tarkemmin liitteessä 2. Analysoinnin perusteella valitut kuusi järjestelmän kannalta tärkeintä laatutekijää ovat alla olevassa listassa tärkeysjärjestyksessä.

1) Testattavuus
2) Muutettavuus
3) Analysoitavuus
4) Ositettavuus
5) Yhteentoimivuus
6) Saavutettavuus

## 6.2 Laadullisten vaatimusten tarkentaminen

Laatutekijät ovat arkkitehtuurin arvioinnin perusta, mutta laatutekijät yksinään eivät kerro riittävän yksiselitteisesti mihin järjestelmän osaan tai käyttötilanteeseen laadullinen vaatimus kohdistuu [CKK01].

ATAM-menetelmässä laadullisia vaatimuksia tarkennetaan sitomalla vaatimukset rajattuun kontekstiin. Kunkin vaatimuksen konteksti kuvataan skenaariolla, joka sisältää laatutekijän lisäksi informaatiota siitä käyttötilanteesta, jossa laatutekijän on toteutettava

asetetut raja-arvot. Skenaarioiden lisäksi määriteltiin jokaiselle laatutekijälle yksi tai useampia tarkennuksia. Laatutekijöiden tarkennukset määrittelevät mihin järjestelmän osa-alueeseen, ominaisuuteen tai toiminnallisuuteen skenaarioina kuvatut laadulliset vaatimukset kohdistuvat.

Esimerkkitapauksen projektissa skenaariot laadittiin ATAM-menetelmän mukaisesti kahdessa vaiheessa. Ensimmäiseen vaiheeseen osallistui pelkästään korvaavan järjestelmän suunnitteluun ja kehitystyöhön osallistuvista henkilöistä koottu arviointiryhmä. Toiseen vaiheeseen osallistuivat ensimmäisen vaiheen arviointiryhmän lisäksi yrityksen johdon edustaja sekä vanhan järjestelmän päävastuulliset suunnittelijat.

## 6.3 Tärkeimmät laadullisia vaatimuksia kuvaavat skenaariot

Laadullisten vaatimusten tarkentaminen tuotti yhteensä 49 skenaariota, jotka järjestettiin tärkeysjärjestykseen kahden tekijän perusteella. Ensimmäinen tekijä kuvaa skenaarion merkitystä koko ohjelmistokehitysprojektin ja sen lopputuloksena syntyvän järjestelmän onnistumisen kannalta. Toinen tekijä kuvaa skenaarion mukaisen laadullisen vaatimuksen saavuttamista tukevan teknisen ratkaisun toteutustyön haastavuutta. Molemmille tekijöille annettiin arvo kolmiportaisella asteikolla (10/20/30). Arkkitehtuurin soveltuvuuden arviointiin päätettiin ottaa mukaan ainoastaan ne skenaariot, joissa toinen tekijä sai asteikon suurimman arvon (30) ja toinen tekijä vähintään toiseksi suurimman arvon (20). Tällaisia skenaariota oli yhteensä 11 kappaletta. Muut skenaariot jätettiin arvioinnin ulkopuolelle.

Arkkitehtuurin soveltuvuuden arvioinnissa käytetyt skenaariot ovat taulukkomuodossa seuraavassa luvussa ja ATAM-menetelmän mukaisena laatupuuna liitteessä 3. Kaikki arviointiryhmän laatimat skenaariot ovat taulukkomuodossa liitteessä 4.

### 6.3.1 Tärkeimmät skenaariot

Seuraava taulukko sisältää arkkitehtuurin soveltuvuuden arvioinnissa käytetyt skenaariot laskevassa tärkeysjärjestyksessä. Jokaisen skenaarion kohdalla on kerrottu taulukon ensimmäisen rivin selitteiden mukaisesti, mitä laatutekijää ja mitä laatutekijän tarkennusta skenaario kuvaa. Lisäksi taulukko sisältää skenaarion arvioitua merkitystä kuvaavat arvot, sekä skenaarion sijoituksen taulukossa.

| # | **Laatutekijä**<br>*Laatutekijän tarkennus*<br>Skenaario | **Arvio** |
|---|---|---|
| **1** | **Muutettavuus**<br>*Ulkoisten järjestelmien käyttäminen*<br>Synkronointilogiikan tiedonlähteenä käytettävä vanhan järjestelmän tietokannan taulu muuttuu. Korvaavan järjestelmän synkronointilogiikan muuttaminen saa kestää enimmillään yhden henkilötyöpäivän. | 30, 30 |
| **2** | **Analysoitavuus**<br>*Aiempaa toteutusta vastaavan uuden toiminnallisuuden toteuttaminen*<br>Järjestelmään lisätään uusi www-sivu, joka sisältää sekä tavallisia, että AJAX -kutsuja. Toteutustyö saa kestää korkeintaan 2 henkilötyöpäivää. | 30, 30 |
| **3** | **Yhteentoimivuus**<br>*Vanhan järjestelmän tietokannan tietojen käyttäminen*<br>Kun käyttäjän pyynnön käsittelyssä tarvittavat tietokannan tiedot päivitetään automaattisesti vanhan järjestelmän tietokannasta, vastaus käyttäjälle täytyy toimittaa alle 5 sekunnissa riippumatta päivitettävän tiedon määrästä. | 30, 30 |
| **4** | **Saavutettavuus**<br>*Tuki virhetilanteiden jäljittämiselle*<br>Järjestelmän on tallennettava tietoa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista. Jokainen pyyntö-vastauspari on tallennettava ainakin jossain vaiheessa niiden sisältämän tiedon käsittelyä varten. | 30, 30 |
| **5** | **Testattavuus**<br>*Järjestelmä tukee yksikkötestien toteuttamista*<br>90 % kaikesta liiketoimintalogiikasta on oltava yksikkötestattavissa. | 30, 20 |

| | | |
|---|---|---|
| **6** | **Testattavuus**<br>*Järjestelmä tukee yksikkötestien toteuttamista*<br>Kehittäjä toteuttaa uudelle toiminnallisuudelle kattavat yksikkötestit. Uusien testien toteuttaminen ei saa vaatia muutoksia aiemmin toteutettuihin testeihin. | 30, 20 |
| **7** | **Analysoitavuus**<br>*Virheellisen tai epäoptimaalisen toteutuksen paikallistaminen*<br>Havaitaan, että järjestelmä laskee käyttäjälle näytettävän arvon väärin. Virheen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 3 tuntia siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 30, 20 |
| **8** | **Ositettavuus**<br>*Järjestelmän kehittäminen ilman riippuvuuksia ulkoisiin järjestelmiin*<br>Ulkoiseen järjestelmään tulevat muutokset saavat aiheuttaa muutoksia vain rajattuun joukkoon korvaavan järjestelmän komponentteja. | 30, 20 |
| **9** | **Yhteentoimivuus**<br>*Riippumattomuus virheistä ulkoisten järjestelmien toiminnassa*<br>Ulkoisen järjestelmän käytön estyminen ei saa johtaa tunnistamattomien tai hallitsemattomien vikojen esiintymiseen järjestelmässä. | 30, 20 |
| **10** | **Analysoitavuus**<br>*Virheellisen tai epäoptimaalisen toteutuksen paikallistaminen*<br>Havaitaan, että erään tiedon hakeminen käyttöliittymään kestää 15 sekuntia. Viiveen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 1 henkilötyöpäivä siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 20, 30 |
| **11** | **Saavutettavuus**<br>*Tuki virhetilanteiden jäljittämiselle*<br>Tuotantosovellusta on pystyttävä suorittamaan testiympäristössä, jossa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista tallennetaan samat tiedot kuin tuotantoympäristössä. | 20, 30 |

# 7 Korvaavan järjestelmän ohjelmistoarkkitehtuuri

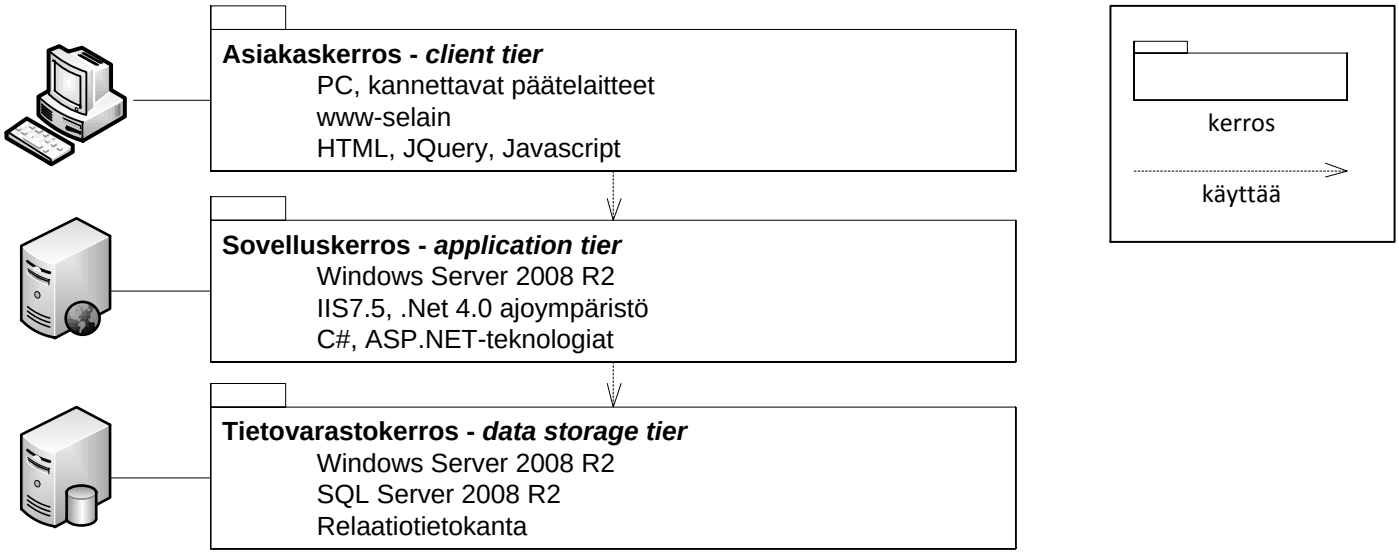
## 7.1 Ohjelmistoarkkitehtuurin dokumentointi

Ohjelmistoarkkitehtuurin sisältämien ratkaisujen selkeä ja ymmärrettävä esittäminen projektin eri sidosryhmille on kriittinen tekijä arkkitehtuuriarvioinnin onnistumiseksi [BZJ04]. Arkkitehtuuridokumentaation ymmärrettävyyteen vaikuttavat esimerkiksi käytettävä merkintätapa ja abstraktiotaso. Esimerkkitapauksessa arkkitehtuurikuvaukset laadittiin käyttäen etupäässä UML -mallinnuskieltä [PiP05].

Arkkitehtuurinäkymien käyttöä pidetään erittäin tärkeänä menetelmänä arkkitehtuureja kuvattaessa [BZJ04]. Näkymien avulla arkkitehtuuria voidaan kuvata useista näkökulmista, eri sidosryhmien tarpeiden mukaisesti. Selkeään kuvaukseen vaadittavien arkkitehtuurinäkymien lukumäärästä ja luonteesta on useita toisistaan poikkeavia näkemyksiä. Esimerkkitapauksen arkkitehtuurikuvaukset ja niissä käytettävät arkkitehtuurinäkymät on valittu ensisijaisesti sillä perusteella, kuinka hyvin ne tukevat tärkeimpien laatuskenaarioiden esittelemistä.

## 7.2 Korvaavan järjestelmän suoritusympäristö

Korvaavan järjestelmän laitteistotason toteutus noudattaa kuvan 13 mukaista kolmikerrosarkkitehtuuria (3-tier architecture) jakautuen kolmeen erilliseen fyysiseen kerrokseen. Fyysisten kerrosten väliset kutsut tapahtuvat aina ylemmästä kerroksesta seuraavana olevaan alempaan kerrokseen.




*Kuva 13: Järjestelmän jakautuminen fyysisiin kerroksiin*

Järjestelmän sovelluskerroksen tarjoamat palvelut ovat käytettävissä pelkästään julkisen internetin kautta, jolloin asiakaskerros muodostuu käyttäjien www-selaimista. Selaimet ovat yleensä pöytätietokoneissa, mutta osa järjestelmän tarjoamista palveluista on suunniteltu käytettäväksi myös kannettavien päätelaitteiden www-selaimilla. Tiedonsiirto asiakaskerroksen ja sovelluskerroksen välillä noudattaa salattua http-protokollaa.

Sovelluskerroksen laitteistotason suoritusympäristönä on yksi fyysinen palvelin, jonka käyttöjärjestelmänä on Microsoft Windows Server 2008 R2. Ohjelmistotason suoritusympäristön muodostavat käyttöjärjestelmän sisältämät IIS 7.5 www-palvelin sekä .Net 4.0 -ohjelmistokehyksen mukainen virtuaalikone. Sovelluskerroksen sovellukset toteutetaan käyttäen C# 4.0 -ohjelmointikieltä sekä Microsoft ASP.NET-teknologioita.

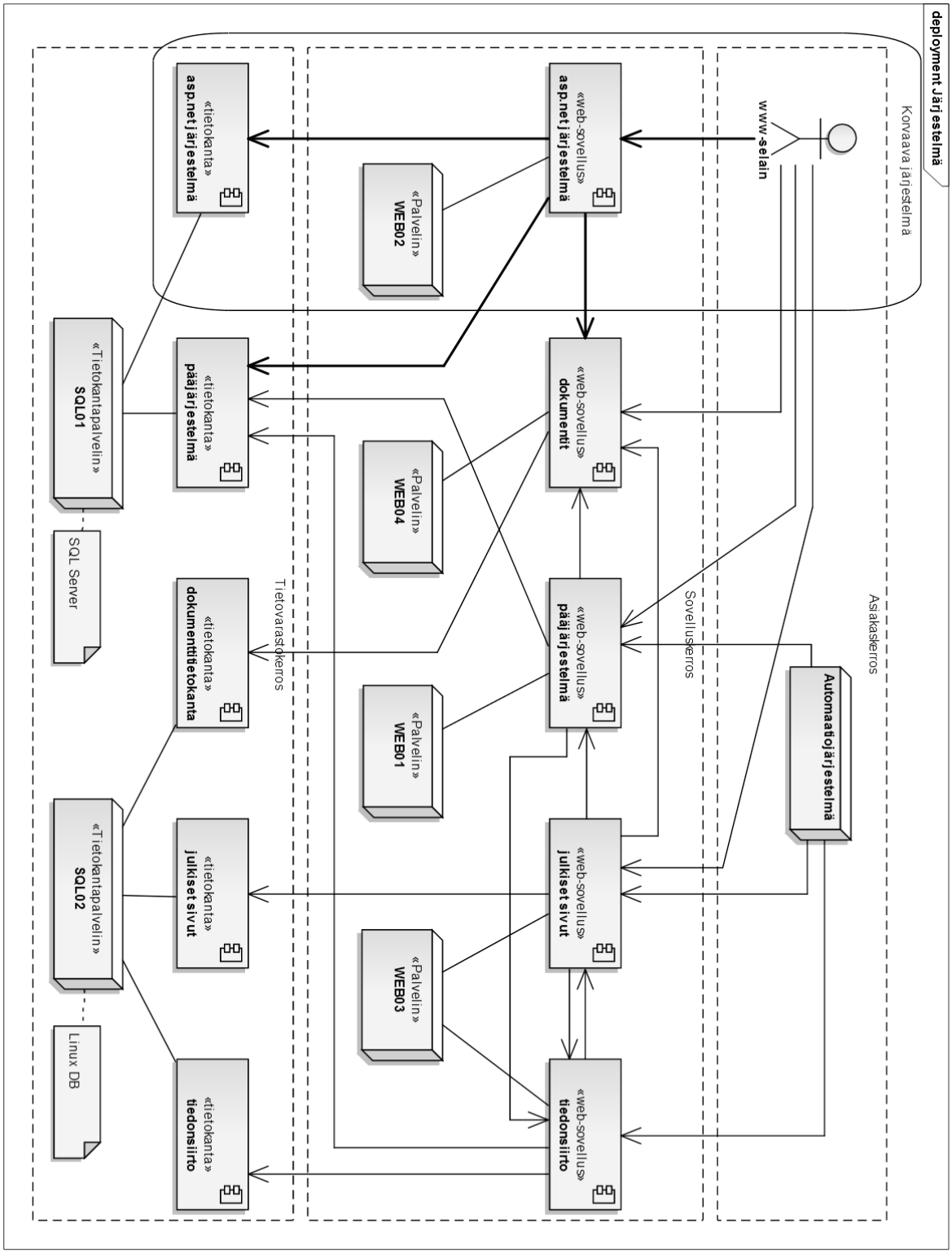
Korvaavan järjestelmän tietovarasto on Microsoft SQL Server 2008 R2 relaatiotietokanta. Tietokantaohjelmistoa suoritetaan fyysisellä palvelimella, jonka käyttöjärjestelmänä on Microsoft Windows Server 2008 R2.

## 7.3 Kokonaisjärjestelmän rakenne vaiheittaisen siirron aikana

Kuvan 14 kaavio esittää vaiheittaisen siirron aikana (luku 4.2.2) käytössä olevan kokonaisjärjestelmän rakenteen laitteisto- ja osajärjestelmätasolla. Kyseessä on siis vanhan järjestelmän ja korvaavan järjestelmän yhdistelmä. Korvaavan järjestelmän komponentit näkyvät kuvan 14 kaaviossa kehystettynä. Muut kaavion 14 kuvaamat komponentit kuuluvat vanhaan järjestelmään.

Järjestelmien yhdistelmä jakaantuu korvaavan järjestelmän tavoin kolmeen kerrokseen: asiakas-, sovellus- ja tietovarastokerrokseen. Siirron aikana kokonaisjärjestelmän asiakaskerroksessa näkyvä käyttöliittymä koostuu osittain vanhan järjestelmän ja osittain korvaavan järjestelmän komponenttien tuottamasta sisällöstä. Tavoitteena on kuitenkin tarjota käyttäjille järjestelmän siirron aikana mahdollisimman yhtenäinen käyttökokemus, vaikka asiakaskerrokseen tuotetaan sisältöä eri teknologioilla. Kaaviossa 14 kuvattu asiakaskerroksen automaatiojärjestelmä käyttää pelkästään vanhan järjestelmän palveluja eikä vaikuta korvaavan järjestelmän toteutukseen.

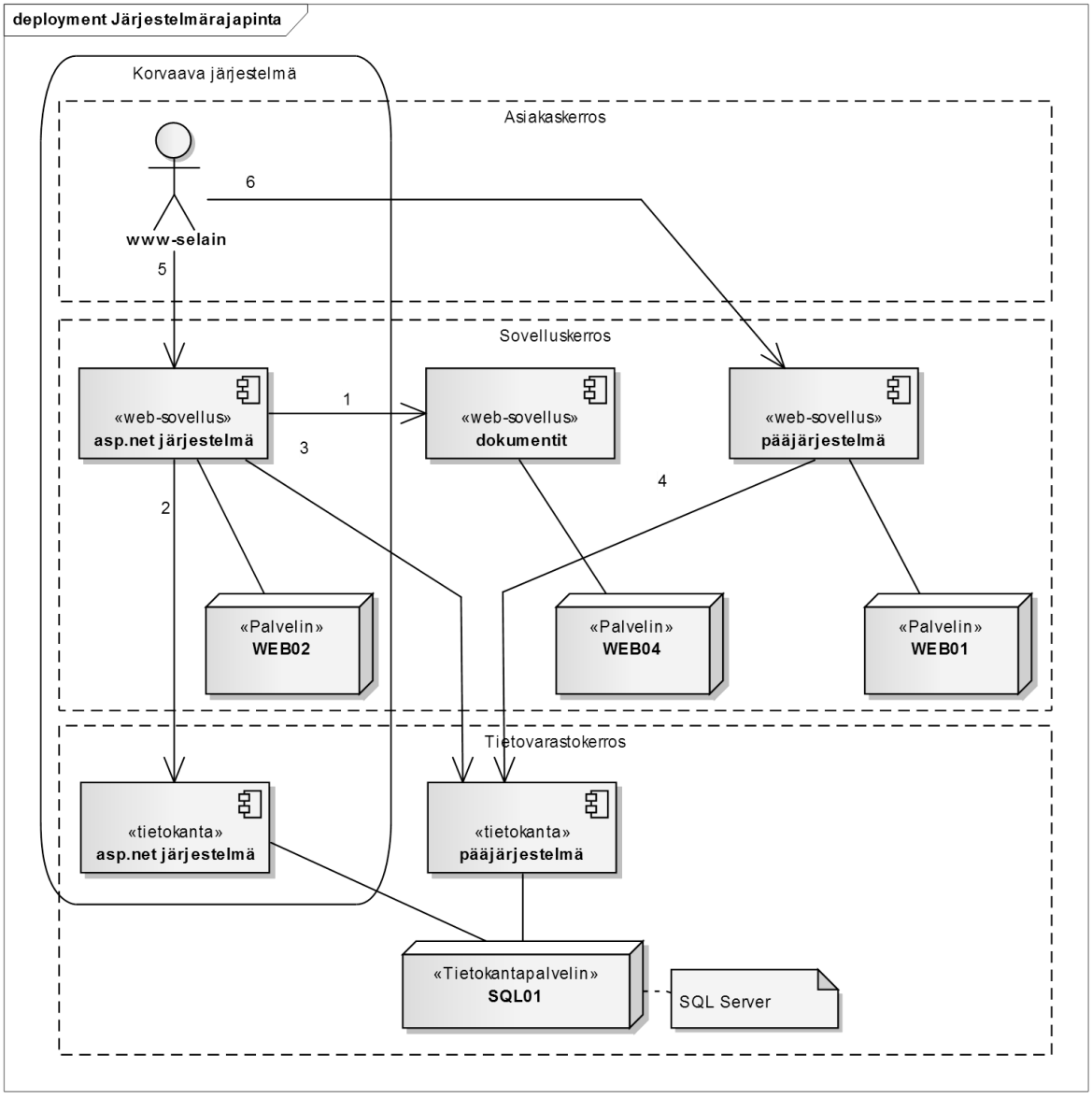
Kokonaisjärjestelmän sovelluskerroksen pääkomponentit ovat web-sovelluksia, joita suoritetaan yhteensä neljällä fyysisellä palvelimella. Korvaavaan järjestelmään kuuluva asp.net-järjestelmä käyttää oman tietokantansa lisäksi vanhaan järjestelmään kuuluvan pääjärjestelmän tietokantaa.




*Kuva 14: Järjestelmän rakenne vaiheittaisen siirron aikana*

Tietovarastokerroksessa järjestelmien välillä ei ole riippuvuuksia. Vanhan järjestelmän ja korvaavan järjestelmän tietokannat ovat samalla tietokantapalvelimella, mutta tietokannat ovat toisistaan riippumattomia.

Kuvan 15 kaaviossa on kuvattu korvaavan järjestelmän ja vanhan järjestelmän välinen järjestelmärajapinta. Tarvittavien yhteyksien muodostaminen sekä varsinainen tiedonvälitys ovat useimmiten osa selaimesta saapuvan kutsun käsittelyä. Tietyt järjestelmien välisen tiedonsiirron tapahtumat käynnistetään ajastetusti.




*Kuva 15: Vanhan ja korvaavan järjestelmän välinen järjestelmärajapinta*

Tiedonvälitykseen järjestelmien välillä käytetään kolmea periaatteeltaan toisistaan poikkeavaa tapaa. Kuvan 15 kaaviossa näkyvät nuolet kuvaavat tiedonvälityksessä käytettäviä yhteyksiä. Eri tiedonvälitystapojen käyttötilanteet sekä käytettävät yhteydet on kuvattu alla.

1) Tietoa välitetään pyyntö-vastaus -periaatteella (request-response) kolmessa tilanteessa:
   - Korvaavan järjestelmän web-sovellus hakee tietoa vanhan järjestelmän dokumentit-web-sovelluksesta. (kuva 15: nuoli 1)
   - Korvaavan järjestelmän web-sovellus hakee tietoa korvaavan järjestelmän omasta tietokannasta. (kuva 15: nuoli 2)
   - Korvaavan järjestelmän web-sovellus hakee tietoa vanhan järjestelmän pääjärjestelmä-tietokannasta. (kuva 15: nuoli 3)
2) Tietoa välitetään jaetun tietovaraston kautta (shared repository) yhdessä tilanteessa:
   - Korvaavan järjestelmän web-sovellus vie tietoa vanhan järjestelmän pääjärjestelmä-tietokantaan ja vanhan järjestelmän pääjärjestelmä-web-sovellus hakee tiedon samasta tietokannasta. (kuva 15: nuolet 3 ja 4)
3) Tietoa välitetään selaimen kautta uudelleenohjauksen avulla kahdessa tilanteessa:
   - Korvaavan järjestelmän web-sovellus vie tietoa vanhan järjestelmän pääjärjestelmä-web-sovellukseen. (kuva 15: nuolet 5 ja 6)
   - Vanhan järjestelmän pääjärjestelmä-web-sovellus vie tietoa korvaavan järjestelmän web-sovellukseen. (kuva 15: nuolet 5 ja 6)

## 7.4 Korvaavan järjestelmän sovelluskerroksen rakenne

Korvaavan järjestelmän sovelluskerroksen toteutus noudattaa kuvan 16 kaavion mukaista *kolmikerrosarkkitehtuuria* (3-layer architecture) jakautuen kolmeen erilliseen loogiseen kerrokseen. Loogisten kerrosten väliset kutsut tapahtuvat aina ylemmästä kerroksesta seuraavana olevaan alempaan kerrokseen. Kaaviossa on kuvattu loogisten kerrosten tärkeimmät ohjelmistokomponentit sekä komponenttien väliset suhteet.

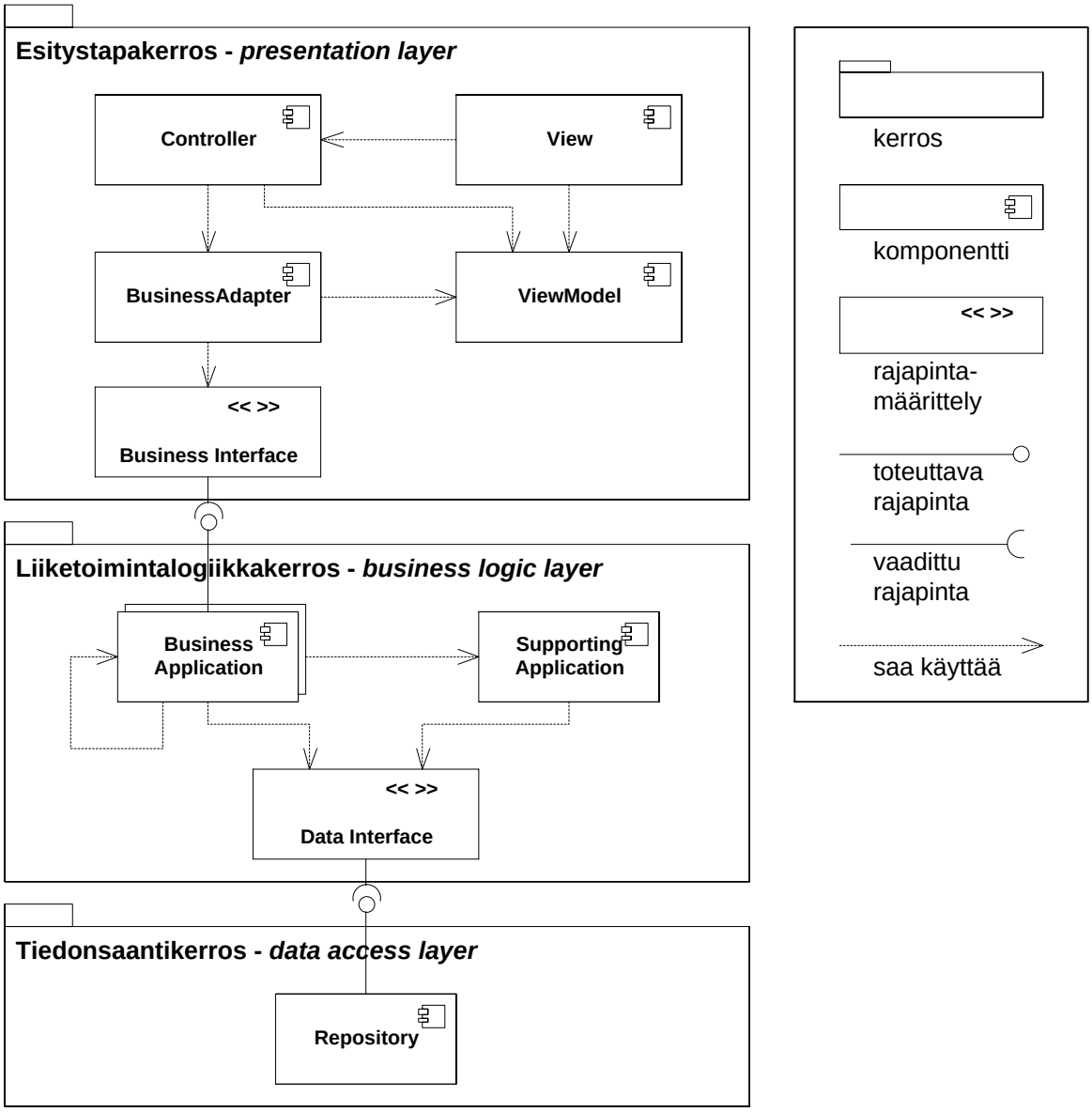
Esitystapakerroksen tehtävänä on tuottaa sisältö käyttäjän www-selaimessa suoritettavaan käyttöliittymään. Kerroksen sisäinen rakenne toteutetaan noudattaen *Malli-Näkymä-Ohjain*-suunnittelumallia (Model-View-Controller, MVC), jossa erotetaan toisistaan sovelluksen tilaa käsittelevä ja tilan säilyttävä logiikka, syötteen käsittely, sekä

käyttöliittymän esittäminen. Toteutuksessa käytetään Microsoftin ASP.NET MVC -ohjelmistokehystä.

MVC-suunnittelumallin mukaan *malli*-komponentti on vastuussa sekä tiedon säilyttämisestä että tietoa käyttävien komponenttien informoimisesta kun mallin sisältämä tieto muuttuu. ASP.NET MVC -suunnittelumallin mukaan malli-komponentti on kuitenkin vain tiedon säilyttäjä, eikä sillä ole aktiivisen informoijan roolia. Korvaavassa järjestelmässä tiedon käsittely tapahtuu liiketoimintalogiikkakerroksessa ja tiedon säilyttäminen tiedonsaantikerroksen kautta käytettävässä tietovarastokerroksessa. ASP.NET MVC -suunnittelumallin mukaista malli-komponenttia vastaa kuvan 16 *BusinessAdapter*, joka tarjoaa muille esitystapakerroksen komponenteille pääsyn järjestelmän sisältämään tietoon *Business Interface* -rajapinnan kautta. MVC-mallin ohjainta vastaa kuvan 16 *Controller*-komponentti ja näkymää *View*-komponentti. Kaaviossa esitetty *ViewModel*-komponentti on tiedonsiirto-objekti, jota ohjain, malli ja näkymä käyttävät keskinäiseen tiedonvälitykseen.

Liiketoimintalogiikkakerroksen komponenttien vastuulla on kaikki järjestelmän sisältämä toiminnallisuus, joka ei liity käyttöliittymän tuottamiseen eikä tiedon varastointiin. Kuvan 16 *Business Application*-komponentit tarjoavat palveluja käyttöliittymäkerrokselle ja käyttävät *Supporting Application* -komponenttien tarjoamia palveluja. Kerroksen komponentit käyttävät tiedonsaantikerroksen palveluja *Data Interface* -rajapinnan kautta.

Tiedonsaantikerroksen tehtävänä on käsitellä järjestelmän tietovarastona toimivan relaatiotietokannan tietoja. Liiketoimintalogiikkakerros käyttää tietoja objektimuotoisena ja tiedonsaantikerroksen tehtävänä on muuttaa relaatiomuotoinen tieto objektimuotoiseksi ja takaisin. Tiedonsaantikerroksen *Repository*-komponentit sisältävät muunnokseen (Object-relational mapping, OR/M) tarvittavan sovelluslogiikan. Komponenttien toteutuksessa on käytetty NHibernate-ohjelmistokehystä [KHB09].




*Kuva 16: Sovelluskerroksen tärkeimmät komponentit ja niiden väliset suhteet*

# 8 Ohjelmistoarkkitehtuurin soveltuvuus

## 8.1 Soveltuvuuden analysointi ja arviointi

Tutkielman esimerkkitapauksessa suunnitellun ohjelmistoarkkitehtuurin soveltuvuus arvioitiin ATAM-menetelmän mukaisesti. Arviointiryhmän kokoamisessa käytetyt periaatteet on kuvattu luvussa 3.2 ja arviointiryhmän kokoonpano luvussa 5.6.

Arkkitehtuurin soveltuvuuden arviointi käsitellään tutkielmassa kahdessa osassa. Ensimmäiseksi analysoidaan arkkitehtonisten lähestymistapojen (katso luku 3.3.1) vaikutuksia tärkeimmiksi arvioituihin laadullisia vaatimuksia kuvaaviin skenaarioihin. Analysoinnin jälkeen arvioidaan, antaako arkkitehtuuri kokonaisuutena riittävät edellytykset järjestelmälle asetettujen laadullisten tavoitteiden toteutumiselle.

Esimerkkitapauksen arvioinnin aikana havaittiin, ettei kaikkien laadullisten vaatimusten toteutumiseen voida merkittävästi vaikuttaa arkkitehtonisilla suunnittelupäätöksillä, vaikka vaatimukset olisi arvioitu järjestelmän kannalta tärkeiksi.

## 8.2 Tärkeimpiin skenaarioihin vaikuttavat arkkitehtoniset lähestymistavat

Tärkeimpien laatuskenaarioiden analysoinnissa selvitetään, mitkä arkkitehtoniset lähestymistavat vaikuttavat skenaarioilla kuvattujen laadullisten vaatimusten toteutumiseen. Arkkitehtoniset lähestymistavat analysoidaan ja samalla selvitetään, sisältävätkö ne käsiteltävän skenaarion kontekstissa mahdollisia riskejä, ei-riskejä, herkkyyspisteitä tai kompromissipisteitä (katso luku 3).

Kunkin skenaarion kuvauksessa kerrotaan, mihin laatutekijään ja mihin laatutekijän tarkennukseen skenaario liittyy ja kuinka tärkeäksi skenaario on ATAM-menetelmän aiempien vaiheiden aikana arvioitu (katso luku 6). Skenaariot käsitellään laskevassa tärkeysjärjestyksessä.

Skenaario 1

| | |
|---|---|
| **Muutettavuus**<br>*Ulkoisten järjestelmien käyttäminen*<br>Synkronointilogiikan tiedonlähteenä käytettävä vanhan järjestelmän tietokannan taulu muuttuu. Korvaavan järjestelmän synkronointilogiikan muuttaminen saa kestää enimmillään yhden henkilötyöpäivän. | 30, 30 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 1.1) Synkronointilogiikka toteutetaan itsenäisenä komponenttikirjastona

Ohjelmistojärjestelmään tehtävät muutokset toteutuvat muutoksina järjestelmän sisältämiin ohjelmistokomponentteihin. Vaikka muutettavien komponenttien lukumäärästä ei voida suoraan päätellä muutoksista aiheutuvaa kokonaiskustannusta, voidaan kuitenkin olettaa, että muutettavien komponenttien lukumäärän pienentäminen

vähentää muutoksien toteuttamiseen kuluvaa aikaa. Muuttamista edellyttävien komponenttien lukumäärä saadaan pienemmäksi antamalla kunkin komponentin vastuulle ainoastaan toisiinsa liittyviä tehtäviä [BCK03].

Skenaarioon liittyvä tiedonsiirtologiikka toteutetaan muusta järjestelmästä riippumattomana komponenttikirjastona, jolloin toteutukseen tehtävät muutokset kohdistuvat selkeästi rajattuun osaan ohjelmistoa. Korvaavan järjestelmän muut osat käyttävät tiedonsiirtokirjaston sisältämää logiikkaa palvelurajapinnan kautta, eikä kirjaston sisäisen toteutuksen muuttaminen edellytä rajapinnan muuttamista. Tiedonsiirtokirjastoon kuuluvien komponenttien vastuulla on pelkästään tietokantojen synkronointiin liittyviä tehtäviä.

Analysoinnissa todettiin, että lähestymistavan mukainen sovelluslogiikan jako selkeän roolin omaaviin komponentteihin tukee merkittävästi skenaarion kuvaaman laatutavoitteen saavuttamista. Lähestymistapa ei sisällä järjestelmän laatuun vaikuttavia riskejä eikä se ole herkkyys- tai kompromissipiste järjestelmän muiden laatutekijöiden suhteen.

### 1.2) Tietokannan käytössä sovelletaan repository-suunnittelumallia

Vanhan järjestelmän tietokannan rakenteeseen tehtävien muutosten välittömät vaikutukset korvaavan järjestelmän tiedonsiirtologiikkaan estetään soveltamalla *repository*-suunnittelumallia [BCK03]. Suunnittelumallissa tietokannan tietoja käytetään aina erityisten repository-objektien kautta. Objektien toteutus sisältää tietokannan rakenteen muuttumisesta aiheutuvien poikkeustilanteiden käsittelyn. Tietokannan muuttumisesta aiheutuvat sovelluslogiikan muutokset kohdistuvat useimmiten pelkästään repository-objekteihin. Relaatiotietokannan tietoja käsitellään objektitasolla NHibernate-ohjelmistokehyksen mukaisina *domain-objekteina* [KHB09].

Valittu lähestymistapa vanhan järjestelmän ja korvaavan järjestelmän tietokantojen synkronointiin arvioitiin toimivaksi sillä edellytyksellä, että synkronointi suoritetaan jatkuvasti useita kertoja tunnissa ja samalla kertaa synkronoitavia tietokantarivejä on enimmillään kymmeniä tuhansia. Riskitekijäksi todettiin vaikeasti ennustettava järjestelmän suorituskykyvaatimusten kasvu synkronoitavien tietokantarivien määrän kasvaessa.

Testattavuuteen liittyväksi riskiksi arvioitiin teknologiaa käyttävän toteutuksen oikeellisen toiminnan vaikea varmentaminen. Analysoitavuuteen liittyväksi riskiksi

todettiin NHibernate-ohjelmistokehyksen käyttöön liittyvien suositeltujen käytäntöjen vähäinen määrä. Lähestymistapa on kompromissipiste järjestelmän suorituskykyyn liittyvien vaatimusten ja järjestelmän muutettavuuteen liittyvien vaatimusten välillä.

Skenaario 2

| | |
|---|---|
| **Analysoitavuus**<br>*Aiempaa toteutusta vastaavan uuden toiminnallisuuden toteuttaminen*<br>Järjestelmään lisätään uusi www-sivu, joka sisältää sekä tavallisia, että AJAX-kutsuja. Toteutustyö saa kestää korkeintaan 2 henkilötyöpäivää. | 30, 30 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 2.1) Järjestelmän rakenne noudattaa 3-kerrosarkkitehtuuria

Ohjelmistojärjestelmän jakaminen loogisiin osiin lisää järjestelmän analysoitavuutta [BCK03]. Analysoitavuuden kasvu on riippuvainen niistä periaatteista, joita noudattaen loogiset osat on muodostettu. Esimerkkitapauksen korvaava järjestelmä on jaettu loogisiin osiin useilla eri abstraktiotasoilla. Useimmiten loogiset osat on muodostettu sijoittamalla samankaltaista tehtävää varten toteutettu sovelluslogiikka samaan loogiseen osaan. Korkeimmalla abstraktiotasolla toteutuva 3-kerrosarkkitehtuuri ja loogisten kerrosten vastuulla olevat tehtävät on kuvattu luvussa 7.2.

Kerrosarkkitehtuurin mukaista rakennetta ei arvioitu riskiksi. Lähestymistavan arvioitiin vaikuttavan heikentävästi järjestelmän suorituskykyyn, mutta negatiivisen vaikutuksen arvioitiin olevan niin pieni, ettei sillä ole järjestelmän käytön kannalta oleellista merkitystä. Vaikutukset analysoitavuuteen arvioitiin positiivisiksi.

### 2.2) Esitystapakerros koostuu selkeän roolin toteuttavista komponenteista

Skenaarion edellyttämien muutosten arvioitiin kohdistuvan pääosin esitystapakerrokseen. Järjestelmän esitystapakerros on jaettu sovellusloogisten vastuiden mukaisesti erillisiin ohjelmistokomponentteihin. Komponenttijako on kuvattu luvussa 7.4.

Arkkitehtuurisuunnitelman mukaisen esitystapakerroksen komponenttijaon arvioitiin parantavan merkittävästi järjestelmän analysoitavuutta, muutettavuutta ja testattavuutta. Valitun lähestymistavan ei arvioitu vaikuttavan heikentävästi yhteenkään arvioinnissa tärkeänä pidettyyn laatutekijään.

### 2.3) Järjestelmän tuottamat www-sivut on määritelty hierarkkisena rakenteena

Suunnitelman mukaan korvaavan järjestelmän toteutuksessa tullaan käyttämään ASP.NET -ohjelmistokehykseen kuuluvia työkaluja ja teknologioita, joiden avulla www-sivuilla usein toistuvat osat voidaan toteuttaa uudelleenkäytettävinä ohjelmisto-komponentteina. Käyttöliittymän toteuttaminen uudelleenkäytettävien komponenttien avulla arvioitiin sekä analysoitavuutta että muutettavuutta parantavaksi ratkaisuksi. Toteutustavan ei arvioitu vaikuttavan heikentävästi yhteenkään arvioinnissa tärkeänä pidettyyn laatutekijään.

### 2.4) Tiedonsaantikerroksen tuottamia domain-objekteja käytetään myös esitystapakerroksessa

NHibernate-ohjelmistokehyksen mukaisia, relaatiotietokannasta haettua tietoa sisältäviä domain-objekteja käytetään tiedon säilyttämiseen ja välittämiseen korvaavan järjestelmän kaikissa loogisissa kerroksissa. Valitulla lähestymistavalla todettiin olevan positiivinen vaikutus järjestelmän analysoitavuuteen, koska tietoa välittävien objektien toteutus on kaikkialla samanlainen.

Järjestelmässä kaikki yhden http-pyynnön käsittelyssä käytettävät tiedot noudetaan tietokannasta saman tietokantatransaktion sisällä. Transaktio aloitetaan ensimmäisen tietokantakyselyn alkaessa ja lopetetaan http-pyynnön käsittelyn päättyessä. Kyselyihin ja tietokantayhteyksien hallintaan liittyvä sovelluslogiikka sijaitsee järjestelmän tiedonsaantikerroksessa, josta tiedot toimitetaan ylempiin kerroksiin domain-objekteina. Domain-objektien sisältämät tiedot noudetaan tietokannasta joko ensimmäisen kyselyn yhteydessä tai vasta kun tietoa ensimmäisen kerran luetaan objektista. Muutokset objektien sisältämään tietoon viedään lopullisesti tietokantaan tietokantatransaktion päättyessä. Järjestelmän kehittäjien täytyy ottaa huomioon domain-objektien sisältämien tietojen kytkös tietokantaan ja transaktioon riippumatta siitä missä järjestelmän loogisessa kerroksessa objekteja käytetään.

Edellä kuvattu kytkös tiedonsaantikerroksen ja esitystapakerroksen välillä arvioitiin analysoitavuutta merkittävästi heikentäväksi tekijäksi. Arviointiryhmän mukaan valitun lähestymistavan negatiiviset vaikutukset järjestelmän analysoitavuuteen ovat positiivisia vaikutuksia suuremmat. Lähestymistavan vaikutukset muihin laatutekijöihin arvioitiin vähäisiksi.

### 2.5) Järjestelmän toteutuksessa käytetään mahdollisimman paljon valmiita väliohjelmistopalveluja

Korvaavan järjestelmän suoritusympäristöön (ks. luku 7.2) kuuluvat palvelinohjelmistot ja järjestelmässä käytetyt ohjelmistokehykset tarjoavat runsaasti väliohjelmistopalveluja eri käyttötarkoituksiin. Useat tässä skenaariossa välttämättömät toiminnot sisältyvät näihin väliohjelmistopalveluihin.

Väliohjelmistopalvelujen käytön arvioitiin selkeyttävän ohjelmiston rakennetta, parantavan analysoitavuutta ja nopeuttavan ohjelmistokehitystä. Väliohjelmistopalvelujen käytöllä ei arvioitu olevan negatiivisia vaikutuksia järjestelmän muihin laatutekijöihin.

## Skenaario 3

| | |
|---|---|
| **Yhteentoimivuus**<br>*Vanhan järjestelmän tietokannan tietojen käyttäminen*<br>Kun käyttäjän pyynnön käsittelyssä tarvittavat tietokannan tiedot päivitetään automaattisesti vanhan järjestelmän tietokannasta, vastaus käyttäjälle täytyy toimittaa alle 5 sekunnissa riippumatta päivitettävän tiedon määrästä. | 30, 30 |

## Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 3.1) Vanha ja korvaava järjestelmä käyttävät samaa tietokantapalvelinta

Vanha järjestelmä ja korvaava järjestelmä käyttävät tiedon säilyttämiseen kahta toisistaan riippumatonta relaatiotietokantaa. Tietokannat on asennettu samaan, yhdellä fyysisellä palvelimella suoritettavaan tietokantapalvelinohjelmistoon (katso kuva 15).

Lähestymistavan arvioitiin parantavan merkittävästi vanhan järjestelmän ja korvaavan järjestelmän yhteentoimivuutta. Analysoinnissa kiinnitettiin erityistä huomiota järjestelmän suorituskykyyn kohdistuvien negatiivisten vaikutusten havaitsemiseksi. Lähestymistavan ei kuitenkaan arvioitu olevan kompromissipiste minkään laatutekijöiden välillä.

### 3.2) Järjestelmässä käytetään NHibernate-ohjelmistokehystä

Relaatiotietokannan tietoja käsitellään järjestelmän liiketoimintalogiikkakerroksessa NHibernate-ohjelmistokehyksen mukaisena objekteista koostuvana tietorakenteena. Ohjelmistokehys tarjoaa useita vaihtoehtoisia tapoja tietojen siirtoon objektitietorakenteen ja tietokannan välillä.

NHibernate-ohjelmistokehyksen todettiin olevan monipuolisesti muokattavissa ja parantavan merkittävästi järjestelmän muutettavuutta ja analysoitavuutta. Toisaalta kehyksen arvioitiin heikentävän järjestelmän suorituskykyä ja siksi vaikeuttavan skenaarion mukaisen aikavaatimuksen saavuttamista. Ohjelmistokehyksen käytön todettiin olevan järjestelmän herkkyyspiste muutettavuuden, analysoitavuuden, yhteentoimivuuden ja suorituskyvyn suhteen. Lähestymistapaa voidaan pitää myös kompromissipisteenä edellisten laatutekijöiden välillä, koska ohjelmistokehyksen käytöstä aiheutuvat vaikutukset laatutekijöihin eivät ole samansuuntaisia.

Skenaario 4

| | |
|---|---|
| **Saavutettavuus**<br>*Tuki virhetilanteiden jäljittämiselle*<br>Järjestelmän on tallennettava tietoa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista. Jokainen pyyntö-vastauspari on tallennettava ainakin jossain vaiheessa niiden sisältämän tiedon käsittelyä varten. | 30, 30 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 4.1) Järjestelmän sovelluskerroksen rakenne noudattaa 3-kerrosarkkitehtuuria

Sovelluskerroksen rakenne noudattaa luvun 7.4 mukaista 3-kerrosarkkitehtuuria. Kerrosrakenteen vuoksi loogisesti samankaltaiset tehtävät suoritetaan samassa järjestelmän osassa ja tehtävien suorittamisessa käytetyt ja tuotetut tiedot on helppo tallentaa.

Kerrosarkkitehtuuri helpottaa skenaarion edellyttämän toiminnallisuuden toteuttamista ja vaikuttaa siten positiivisesti järjestelmän saavutettavuuteen. Skenaarion 2 analysoinnin tuottamien tulosten mukaisesti kerrosarkkitehtuuri heikentää järjestelmän suorituskykyä, mutta vaikutuksella ei ole järjestelmän käytön kannalta oleellista merkitystä.

### 4.2) Järjestelmän sovelluskerroksen sovelluslogiikka on toteutettu ohjelmistokomponentteina

Järjestelmän sovelluskerroksen sisältämän sovelluslogiikan jakaantuminen ohjelmistokomponentteihin on kuvattu luvussa 7.4. Komponenttijaon vaikutukset järjestelmän laatutekijöihin arvioitiin vastaaviksi kuin arkkitehtonisen lähestymistavan 4.1 yhteydessä esitellyt kerrosarkkitehtuurin vaikutukset.

### 4.3) Järjestelmän toteutuksessa sovelletaan aspektiperustaista ohjelmointimallia

*Aspektiperustaista ohjelmointimallia* (Aspect-oriented programming, AOP) [HaM05] käyttämällä voidaan ohjelmistojärjestelmään lisätä uusia ohjelmistokomponentteja ilman että järjestelmän rakennetta tai komponenttijakoa muutetaan. Skenaarion edellyttämä sovelluslogiikka voidaan toteuttaa aspektiperustaisen ohjelmointimallin avulla komponenttina, jonka käyttöönotto tai poisto ei edellytä ohjelmiston kääntämistä uudelleen. Järjestelmässä käytettävä aspektiperustainen ohjelmointimalli perustuu *Spring-ohjelmistokehykseen* [HaM05].

Aspektiperustaisen ohjelmointimallin käyttäminen ei arvion mukaan huononna järjestelmän analysoitavuutta. Käytön arvioitiin helpottavan skenaarion edellyttämän toiminnallisuuden toteuttamista ja siten vaikuttavan positiivisesti järjestelmän saavutettavuuteen. Ratkaisun vaikutuksia järjestelmän suorituskykyyn ei pystytty arvioimaan tarkasti, koska skenaarion mukaisen tallennettavan tiedon määrä ei ollut tiedossa arvioinnin aikaan. Mahdollista suorituskykyyn liittyvää riskiä ei kuitenkaan pidetty merkittävänä, koska AOP-komponentit voidaan tarvittaessa poistaa käytöstä kustannustehokkaasti.

Skenaario 5

| | |
|---|---|
| **Testattavuus**<br>*Järjestelmä tukee yksikkötestien toteuttamista*<br>90 % kaikesta liiketoimintalogiikasta on oltava yksikkötestattavissa. | 30, 20 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 5.1) Järjestelmässä käytetään dependency injection -suunnittelumallia

Korvaavan järjestelmän ohjelmistokomponenttien välisten riippuvuuksien hallinnassa noudatetaan *dependency injection* -suunnittelumallia (DI) [HaM05]. Suunnittelumallin mukainen toteutus perustuu Spring-ohjelmistokehyksen käyttöön. DI-suunnittelumalli helpottaa ohjelmistokomponenttien sisältämän sovelluslogiikan testaamista muusta järjestelmästä riippumattomasti. Lähestymistavan arvioitiin parantavan merkittävästi järjestelmän testattavuutta ilman negatiivisia vaikutuksia järjestelmän muihin laatutekijöihin.

### 5.2) Järjestelmän toteutuskieli on C# 4.0

Korvaavan järjestelmän toteutuskieleksi on valittu *C# 4.0* [Tro10], joka on yksi Microsoft .NET 4 -ohjelmistokehykseen sisältyvistä ohjelmointikielistä. C# -kielen ominaisuuksien arvioitiin tukevan järjestelmän testattavuutta merkittävästi. Kielen käytöllä ei arvioitu olevan negatiivista vaikutusta järjestelmän muihin laatutekijöihin.

### 5.3) Järjestelmä on jaettu loogisiin kokonaisuuksiin eri abstraktiotasoilla

Järjestelmä on jaettu loogisiin kokonaisuuksiin erilaisin perustein. Skenaarion kannalta tärkeimmät järjestelmän sisäiset rakenteet on kuvattu luvussa 7.4. Järjestelmän testaus voidaan jakaa osiin kerrosjaon mukaisia loogisia rajoja noudattaen, jolloin testeissä käytettävät syötteet ja odotetut tulokset ovat selkeästi määriteltävissä. Järjestelmän sisäiset ohjelmistorajapinnat helpottavat testattavan sovelluslogiikan eristämistä muusta toteutuksesta. Arkkitehtuurin mukaisen loogisen jaon arvioitiin parantavan järjestelmän testattavuutta ilman merkittäviä negatiivisia vaikutuksia muihin laatutekijöihin.

#### Skenaario 6

| | |
|---|---|
| **Testattavuus**<br>*Järjestelmä tukee yksikkötestien toteuttamista*<br>Kehittäjä toteuttaa uudelle toiminnallisuudelle kattavat yksikkötestit.<br>Uusien testien toteuttaminen ei saa vaatia muutoksia aiemmin toteutettuihin testeihin. | 30, 20 |

#### Skenaarioon liittyvät arkkitehtoniset lähestymistavat

Arviointiryhmän mukaan skenaarion 5 yhteydessä analysoitujen arkkitehtonisten lähestymistapojen vaikutukset ovat yhtäpitävät skenaarion 6 kontekstissa. Muilla lähestymistavoilla ei arvioitu olevan vaikutusta tämän skenaarion edellyttämän teknisen ratkaisun toteuttamiseen.

Skenaario 7

| | |
|---|---|
| **Analysoitavuus**<br>*Virheellisen tai epäoptimaalisen toteutuksen paikallistaminen*<br>Havaitaan, että järjestelmä laskee käyttäjälle näytettävän arvon väärin. Virheen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 3 tuntia siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 30, 20 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

### 7.1) Järjestelmä on jaettu loogisiin kokonaisuuksiin eri abstraktiotasoilla

Ohjelmistojärjestelmän jakaminen loogisiin osiin lisää järjestelmän analysoitavuutta [BCK03]. Korvaavan järjestelmän jako loogisiin kokonaisuuksiin perustuu useimmiten ohjelmistokomponenteille määriteltyihin rooleihin. Skenaarion kannalta tärkeimmät korvaavan järjestelmän sisäiset rakenteet on kuvattu luvussa 7.4.

Voidaan olettaa, että ainakin osasta järjestelmän toiminnassa havaituista virheistä voidaan päätellä minkä tyyppinen sovelluslogiikka on aiheuttanut virhetilanteen. Tällä oletuksella rooliperustainen komponenttijako helpottaa vikojen löytämistä ja parantaa järjestelmän analysoitavuutta. Arviointiryhmä piti arkkitehtuurin mukaisen komponenttijaon vaikutusta järjestelmän analysoitavuuteen positiivisena. Vaikutusta ei pidetty erityisen merkittävänä.

### 7.2) Käytetyt teknologiat ja kehitysympäristö mahdollistavat järjestelmässä suoritettavien testien etenemisen seuraamisen

Korvaava järjestelmä on suunniteltu toteutettavaksi MS .NET 4.0 -ohjelmistokehyksen teknologioilla. Kehitysympäristönä on MS Visual Studio 2010, joka yhdessä toteutukseen käytettävän C# -ohjelmointikielen kanssa mahdollistaa ohjelmistokoodin suorittamisen kehitysympäristön sisällä. Suorituksen etenemistä koodissa voidaan seurata rivi riviltä ja muuttujien saamat arvot voidaan tarkistaa ja tarvittaessa myös muuttaa suorituksen aikana.

MS .NET 4.0 -teknologioiden kanssa käytettävissä olevien työkalujen arvioitiin tukevan järjestelmän testattavuutta merkittävästi. Valittujen teknologioiden ja niihin liittyvien työkalujen ei arvioitu vaikuttavan negatiivisesti järjestelmän muihin laatutekijöihin.

Skenaario 8

| | |
|---|---|
| **Ositettavuus**<br>*Järjestelmän kehittäminen ilman riippuvuuksia ulkoisiin järjestelmiin*<br>Ulkoiseen järjestelmään tulevat muutokset saavat aiheuttaa muutoksia vain rajattuun joukkoon järjestelmän komponentteja. | 30, 20 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

**8.1) Ulkoisten järjestelmien kanssa kommunikoiva sovelluslogiikka toteutetaan muusta järjestelmästä riippumattomina komponentteina**

Ulkoisten järjestelmien kanssa kommunikoivat korvaavan järjestelmän komponentit sisältävät palvelurajapinnan, jonka kautta muut järjestelmän komponentit käyttävät ulkoisten järjestelmien palveluja. Tällöin ulkoiseen järjestelmään tulevat muutokset edellyttävät muutoksia ainoastaan palvelurajapinnan toteuttavaan komponenttiin.

Valittu lähestymistapa toteuttaa suoraan skenaarion kuvaamat laadulliset vaatimukset ja parantaa järjestelmän ositettavuutta. Arvion mukaan lähestymistavalla ei ole vaikutuksia järjestelmän muihin laatutekijöihin.

Skenaario 9

| | |
|---|---|
| **Yhteentoimivuus**<br>*Riippumattomuus virheistä ulkoisten järjestelmien toiminnassa*<br>Ulkoisen järjestelmän käytön estyminen ei saa johtaa tunnistamattomien tai hallitsemattomien vikojen esiintymiseen järjestelmässä. | 30, 20 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

Arviointiryhmä päätyi analysoinnin perusteella siihen lopputulokseen, että suunnitellut arkkitehtoniset lähestymistavat eivät vaikuta tämän skenaarion edellyttämän teknisen ratkaisun toteuttamiseen. Arkkitehtonisen lähestymistavan 8.1 yhteydessä on kuvattu ulkoisten järjestelmien kanssa kommunikoivien ohjelmistokomponenttien toimintaa, mutta kuvauksesta puuttuu ulkoisista järjestelmistä johtuvien virhetilanteiden käsittelylogiikka. Arviointiryhmän mukaan arkkitehtuurisuunnitelmaa tulee laajentaa siten, että tässä skenaariossa kuvattujen laadullisten vaatimusten saavuttamiseen esitetään ratkaisu.

Skenaario 10

| | |
|---|---|
| **Analysoitavuus**<br>*Virheellisen tai epäoptimaalisen toteutuksen paikallistaminen*<br>Havaitaan, että erään tiedon hakeminen käyttöliittymään kestää 15 sekuntia. Viiveen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 1 henkilötyöpäivä siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 20, 30 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

Arviointiryhmän mukaan skenaarion 7 yhteydessä analysoitujen arkkitehtonisten lähestymistapojen vaikutukset ovat yhtäpitävät skenaarion 10 kontekstissa. Muilla lähestymistavoilla ei arvioitu olevan vaikutusta tämän skenaarion edellyttämän teknisen ratkaisun toteuttamiseen.

Skenaario 11

| | |
|---|---|
| **Saavutettavuus**<br>*Tuki virhetilanteiden jäljittämiselle*<br>Tuotantosovellusta on pystyttävä suorittamaan testiympäristössä, jossa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista tallennetaan samat tiedot kuin tuotantoympäristössä. | 20, 30 |

Skenaarioon liittyvät arkkitehtoniset lähestymistavat

Arviointiryhmä päätyi analysoinnin perusteella siihen lopputulokseen, että suunnitellut arkkitehtoniset lähestymistavat eivät vaikuta tämän skenaarion edellyttämän teknisen ratkaisun toteuttamiseen.

## 8.3 Korvaavan järjestelmän ohjelmistoarkkitehtuurin soveltuvuus

Merkittävimpien ohjelmistoarkkitehtuurissa havaittujen riskien todettiin liittyvän NHibernate-ohjelmistokehyksen käyttöön. Laadullisten skenaarioiden analysoinnissa ohjelmistokehyksen havaittiin vaikuttavan negatiivisesti useisiin eri laatutekijöihin. Kokonaisuutena vaikutukset arvioitiin kuitenkin järjestelmän laatua parantavaksi ja kehystä päätettiin käyttää korvaavan järjestelmän toteutuksessa.

NHibernate-ohjelmistokehyksen mukaisten domain-objektien käyttö järjestelmän sovelluskerroksen kaikissa loogisissa kerroksissa arvioitiin niin suureksi riskiksi, että

lähestymistavasta päätettiin luopua. Korvaavana ratkaisuna domain-objektien käyttö päätettiin rajoittaa järjestelmän tiedonsaanti- ja liiketoimintalogiikkakerroksiin. Tiedonsiirrossa liiketoimintalogiikka- ja esitystapakerroksen välillä päätettiin käyttää objekteja, jotka ovat täysin riippumattomia tiedonsaantikerroksesta ja tietokannasta.

Järjestelmän komponenttijakoon liittyvien arkkitehtonisten lähestymistapojen todettiin vaikuttavan positiivisesti useisiin tärkeiksi arvioituihin laatutekijöihin. Tietyille ohjelmistokomponenteille suunnitellut sovellusloogiset vastuut todettiin puutteellisesti määritellyiksi.

Kokonaisuutena arkkitehtuurin arvioitiin soveltuvan korvaavan järjestelmän toteutukseen edellä mainituin muutoksin. Arkkitehtuurin arvioitiin tukevan riittävästi tärkeimpien laatutekijöiden toteutumista ja modernisointiprojektin liiketoiminnallisten tavoitteiden saavuttamista.

# 9 Yhteenveto

Yrityksen harjoittaman liiketoiminnan kehittyminen voi johtaa tilanteeseen, jossa liiketoiminnan tukena käytettävä ohjelmistojärjestelmä ei enää täytä sille asetettuja vaatimuksia. Liiketoiminnan kehittyessä myös järjestelmän täytyy kehittyä. Jos järjestelmään vaadittavat muutokset edellyttävät toimenpiteitä, joita ei laajuutensa tai kestonsa vuoksi voi luokitella järjestelmän ylläpidoksi, voidaan vanha järjestelmä modernisoida.

Yksi vaihtoehto ohjelmistojärjestelmän modernisoimiseksi on vanhan järjestelmän korvaaminen kokonaisuudessaan uudella järjestelmällä. Laaja järjestelmä voidaan korvata vaiheittain, jolloin vanha ja korvaava järjestelmä ovat yrityksen käytössä samanaikaisesti. Modernisointiprojektille laaditaan liiketoiminnalliset tavoitteet, joiden saavuttamista modernisoinnin tuloksena syntyvän järjestelmän täytyy tukea riittävällä tasolla. Korvaavalle järjestelmälle määritellään projektin liiketoiminnallisten tavoitteiden pohjalta joukko laadullisia vaatimuksia.

Ohjelmistoarkkitehtuuri määrittää rajat sitä noudattavan järjestelmän ohjelmistotekniselle laadulle. Arkkitehtuuria arvioimalla on mahdollista selvittää minkälaiset mahdollisuudet arkkitehtuuri antaa määritellyt laadulliset vaatimukset täyttävän järjestelmän toteuttamiselle. Ohjelmistoarkkitehtuuri vaikuttaa toteutettavan järjestelmän kautta liiketoiminnallisten tavoitteiden saavuttamiseen. Arkkitehtuurin arviointi parantaa

mahdollisuuksia havaita ja poistaa arkkitehtuurisuunnitelman sisältämiä riskejä ennen järjestelmän toteuttamista. Riskien välttäminen tuottaa yritykselle taloudellista hyötyä ja tekee liiketoiminnallisten tavoitteiden saavuttamisesta todennäköisempää.

Tutkielmassa käsiteltävässä modernisointiprojektissa korvaavan järjestelmän arkkitehtuuria arvioitiin ATAM-menetelmällä. Esimerkkitapauksessa havaittiin, että vanhan järjestelmän modernisointiin liittyvien liiketoiminnallisten vaatimusten toteutumista voidaan tukea korvaavan järjestelmän arkkitehtuurisuunnitelmaa arvioimalla.

ATAM-menetelmän mukaan yksittäiset järjestelmän laatua kuvaavat laatutekijät eivät sellaisenaan ole käytettävissä arkkitehtuurin arviointiin. Menetelmässä käytetään laadullisia vaatimuksia kuvaavia skenaarioita, joilla laatutekijät sidotaan todennäköiseen käyttötilanteeseen. Jos ohjelmistoarkkitehtuurin kuvaamat lähestymistavat täyttävät skenaarioilla esitetyt laadulliset vaatimukset, voidaan todeta, että arkkitehtuuri tukee skenaarioihin liittyvien laatutekijöiden toteutumista järjestelmässä.

Ohjelmistoarkkitehtuurin arviointi ATAM-menetelmällä edellyttää arkkitehtuuria käyttävän ohjelmistojärjestelmän liiketoiminnallisten tavoitteiden kartoittamista. ATAM-menetelmä ei kuitenkaan kerro, kuinka laatutekijöihin liittyvät vaatimukset johdetaan liiketoiminnallisista tavoitteista. Esimerkkitapauksessa modernisointiprojektin liiketoiminnalliset tavoitteet luokiteltiin ja sijoitettiin kaksitasoiseen hierarkiaan ”korkean tason liiketoiminnallisiin tavoitteisiin” ja niiden saavuttamista tukeviin ”tarkennettuihin liiketoiminnallisiin tavoitteisiin”. Luokittelu helpotti merkittävästi korvaavan järjestelmän tärkeimpien laatutekijöiden määrittelyä.

Arkkitehtuuriarvioinnin yhteydessä havaittiin, että arkkitehtuurisuunnitelman sisältämät arkkitehtonisten lähestymistapojen kuvaukset olivat osin puutteellisia. Arviointiprosessin aikana arkkitehtuurisuunnitelman laatu kuitenkin parani merkittävästi. ATAM-menetelmän mukaisen arkkitehtuuriarvioinnin suurimpana ongelmana pidettiin arviointiin osallistuvien sidosryhmien sitouttamista vaadittuihin tehtäviin. Kaikkia arviointiryhmän jäseniä oli haasteellista saada perehtymään huolellisesti arvioinnin kohteena olevaan arkkitehtuurisuunnitelmaan. Yksi suurimpia arvioinnin tuomia hyötyjä oli arkkitehtuurisuunnitelman sisältämän informaation välittyminen kaikille sidosryhmille viimeistään arviointipalaverien aikana.

# 10 Lähteet


BaG04 Babar M. ja Gorton I., A Comparison of Scenario-Based Software Architecture Evaluation Methods. Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC .04), Busan, Korea, marraskuu 2004, sivut 600 - 607.

BCK03 Bass L., Clements P. ja Kazman R., Software Architecture in Practice Second Edition. Addison-Wesley, 2003.

BCM03 Bianchi A., et al., Iterative Reengineering of Legacy Systems. IEEE Transactions on Software Engineering 29, 3 (2003), sivut 225 - 241.

BLB04 Bengtsson P., et al., Architecture-level modifiability analysis (ALMA), Journal of Systems and Software 69, 1-2, (2004), sivut 129 - 147.

BLW97 Bisbal J., et al., An overview of legacy information system migration. Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference, Hong Kong, Kiina, joulukuu 1997, sivut 529 - 530.

BLW99 Bisbal J., et al., Legacy Information Systems: Issues and Directions. IEEE Software 16, 5 (1999), sivut 103 - 111.

BrS95 Brodie M. ja Stonebraker M., Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach. Morgan Kaufmann Publishers, 1995.

BZJ04 Babar M., Zhu L. ja Jeffery R., A Framework for Classifying and Comparing Software Architecture Evaluation Methods. Proceedings of the Australian Software Engineering Conference (ASWEC), Melbourne, Australia, huhtikuu 2004, sivut 309 - 319.

CBB10 Clements P., et al., Documenting Software Architectures Second Edition. Addison-Wesley, 2010.

CKK01 Clements P., Kazman R. ja Klein M., Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, 2001.

CWS00 Comella-Dorda S., et al., A Survey of Black-Box Modernization Approaches for Information Systems. Proceedings of the 16th IEEE International Conference on Software Maintenance (ICSM'00), San Jose, Kalifornia, Yhdysvallat, lokakuu 2000, sivut 173 - 183.
Myös: http://www.sei.cmu.edu/reports/00tn003.pdf [18.3.2012]

CSL09 Chung L., Sampaio J. ja Leite P., On Non-Functional Requirements in Software Engineering. Conceptual Modeling: Foundations and Applications, Springer, 2009, sivut 638 - 653.

DoN02 Dobrica L. ja Niemelä E., A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering 28,7(2002), sivut 638 - 653.

Gar00 Garlan D., Software architecture: a roadmap. Proceedings of the Conference on The Future of Software Engineering (ICSE '00), Limerick, Irlanti, kesäkuu 2000, sivut 91 - 101.

HaM05 Harrop Rob ja Machacek Jan, Pro Spring. Apress, 2005.

IEE98 IEEE, IEEE Standard for a Software Quality Metrics Methodology, IEEE Std 1061-1998, 1998

IEE00 IEEE, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000, 2000

ISO01 International Organization for Standardization, Software engineering - Product quality - Part 1: Quality model, ISO/IEC 9126-1:2001, 2001

ISO11 International Organization for Standardization, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models, ISO/IEC 25010:2011, 2011

KBW94 Kazman R., et al., SAAM: a method for analyzing the properties of software architectures. Proceedings of the 16th International Conference on Software engineering (ICSE '94), Sorrento, Italia, toukokuu 1994, sivut 81 - 90.

KCW00 Kazman R., Carrière S.J. ja Woods S.G., Toward a discipline of scenario-based architectural engineering, Annals of Software Engineering 9, 1-4, (2000), sivut 5 - 33.

KHB09 Kuaté P., et al., NHibernate in Action, Manning Publications 2009.

KKB98 Kazman R., et al., The Architecture Tradeoff Analysis Method. Proceedings of 4th International Conference on Engineering of Complex Computer Systems (ICECCS '98), Monterey, Kanada, elokuu 1998, sivut 68 - 78.

KKC00 Kazman R., Klein M. ja Clements P., ATAM: Method for Architecture Evaluation, The Carnegie Mellon Software Engineering Institute (SEI) Technical Report CMU/SEI-2000-TR-004, syyskuu 2000. http://www.sei.cmu.edu/reports/00tr004.pdf [18.3.2012]

MCY99 Mylopolous J., Chung L. ja Yu E., From Object-Oriented to Goal-Oriented Requirements Analysis, Communications of the ACM 42, 1,(1999), sivut 31 - 37.

PiP05 Pilone D. ja Pitman N., UML 2.0 in a nutshell, O'Reilly 2005.

RoG08 Roy B. ja Graham T.C.N., Methods for Evaluating Software Architecture: A Survey, Technical Report No. 2008-545, School of Computing, Queen's University at Kingston, Ontario, Kanada, huhtikuu 2008. http://techreports.cs.queensu.ca/files/2008-545.pdf [18.3.2012]

Tro10 Troelsen A., Pro C# 2010 and the .NET 4 Platform, Apress 2010.

TMD10 Taylor R., Medvidović N. ja Dashofy E., Software Architecture Foundations, Theory, and Practice. John Wiley & Sons, Inc., 2010.

WNS97 Weiderman N., et al., Implications of Distributed Object Technology for Reengineering, The Carnegie Mellon Software Engineering Institute (SEI) Technical Report CMU/SEI-97-TR-005, kesäkuu 1997. http://www.sei.cmu.edu/reports/97tr005.pdf [18.3.2012]

WiS02 Williams L. ja Smith C., PASA$^{SM}$ : A Method for the Performance Assessment of Software Architectures. Proceedings of the 3rd international workshop on Software and performance (WOSP '02), Rooma, Italia, heinäkuu 2002, sivut 179 - 188.

ZhY04 Zhang Z. ja Yang H., Incubating Services in Legacy Systems for Architectural Migration. Proceedings of 11th Asia-Pacific Software Engineering Conference (APSEC'04), Busan, Korea, marras-joulukuu 2004, sivut 196 - 203.

# Liite 1. Liiketoiminnallisten tavoitteiden väliset suhteet

Esimerkkitapauksen modernisointiprojektin liiketoiminnalliset tavoitteet määriteltiin yrityksen johtoryhmässä ennen projektin alkua. Tavoitteet kuvattiin hierarkkisesti kahdessa tasossa: korkean tason liiketoiminnallisina tavoitteina ja tarkennettuina liiketoiminnallisina tavoitteina. Määritellyt korkean tason liiketoiminnalliset tavoitteet on lueteltu alla tärkeysjärjestyksessä.

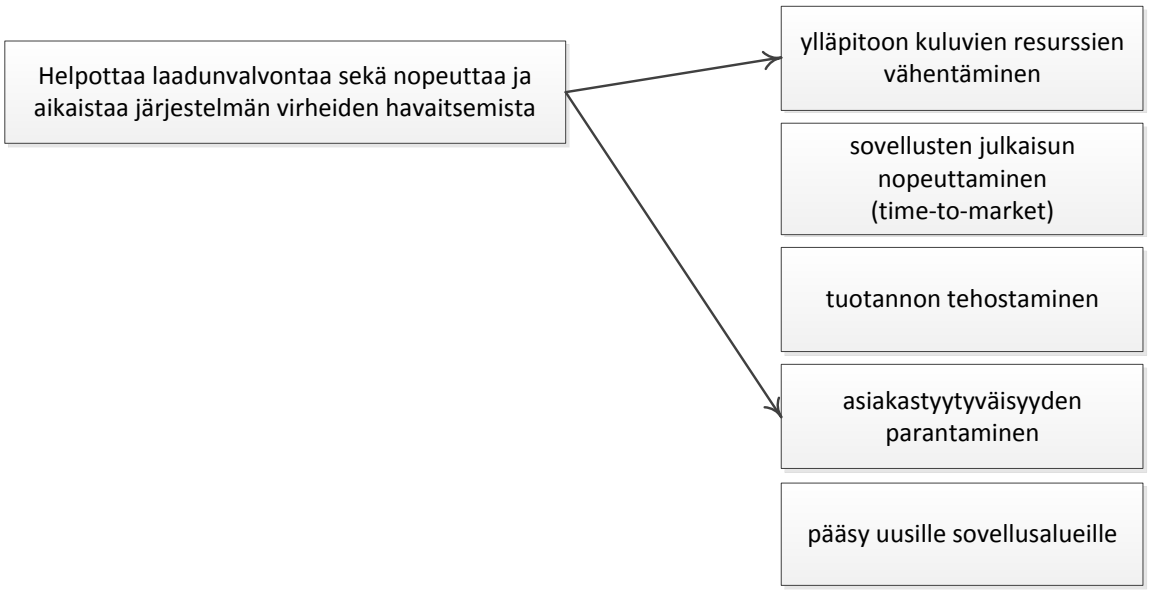
1) Ylläpitoon kuluvien resurssien vähentäminen
2) Sovellusten julkaisun nopeuttaminen (time-to-market)
3) Tuotannon tehostaminen
4) Asiakastyytyväisyyden parantaminen
5) Pääsy uusille sovellusalueille

Korkean tason liiketoiminnalliset tavoitteet todettiin liian abstrakteiksi ja tavoitteita pyrittiin konkretisoimaan laatimalla joukko tarkennettuja liiketoiminnallisia tavoitteita. Tarkennetut liiketoiminnalliset tavoitteet on lueteltu alla tärkeysjärjestyksessä.

1) Helpottaa laadunvalvontaa, sekä nopeuttaa ja aikaistaa järjestelmän virheiden havaitsemista
2) Helpottaa uusien teknologioiden ja innovaatioiden käyttöönottoa järjestelmän kehityksessä
3) Helpottaa valmiiden ratkaisujen hyödyntämistä järjestelmän kehityksessä
4) Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen järjestelmän ominaisuuksiin ja toimintaan sekä yrityksen liiketoiminta-alueeseen
5) Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen käytettäviin teknologioihin ja työkaluihin
6) Työskentelyprosessien uudistaminen
7) Helpottaa uusien henkilöresurssien hankkimista rekrytoimalla
8) Helpottaa ulkopuolisten henkilöresurssien käyttämistä kehitystyössä sekä mahdollistaa järjestelmän kehitysprojektien ulkoistaminen

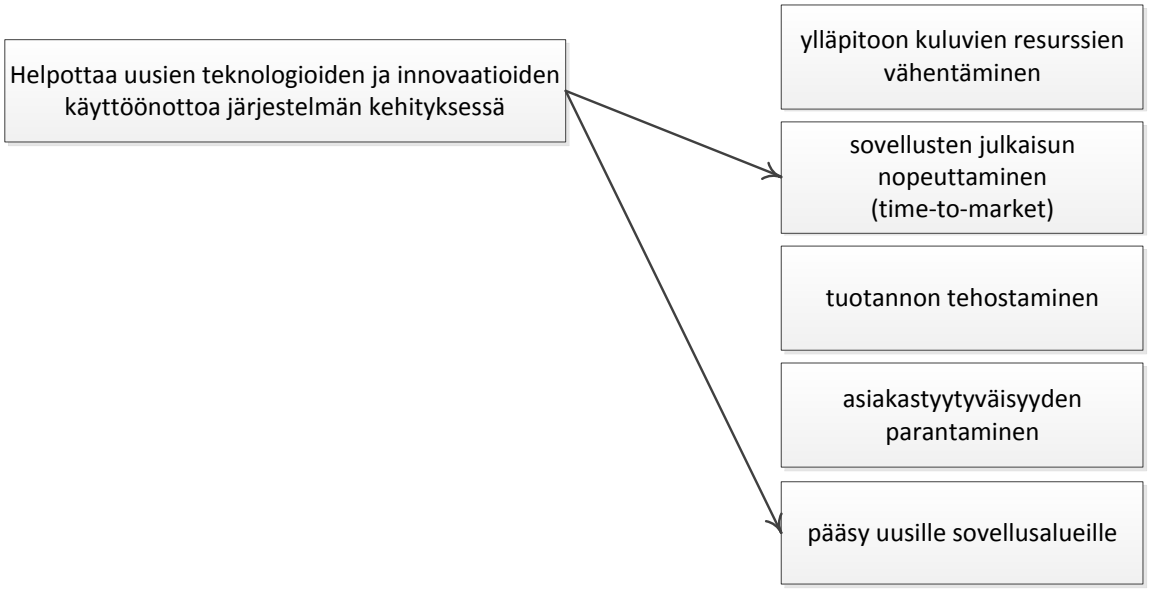
Tarkennetut liiketoiminnalliset tavoitteet johdettiin korkean tason tavoitteista siten, että jokainen tarkennettu tavoite tukee toteutuessaan yhden tai useamman korkean tason tavoitteen toteutumista. Alla on kuvattu yrityksessä laadittu arvio liiketoiminnallisten tavoitteiden välisistä suhteista.

**Helpottaa laadunvalvontaa sekä nopeuttaa ja aikaistaa järjestelmän virheiden havaitsemista**




Tavoitteen toteutumisen arvioitiin vähentävän järjestelmän tuotantoon päätyvien virheiden lukumäärää ja siten vähentävän järjestelmän ylläpitoon kuluvia resursseja. Käyttäjille näkyvien virheiden vähenemisen arvioitiin johtavan asiakastyytyväisyyden paranemiseen.

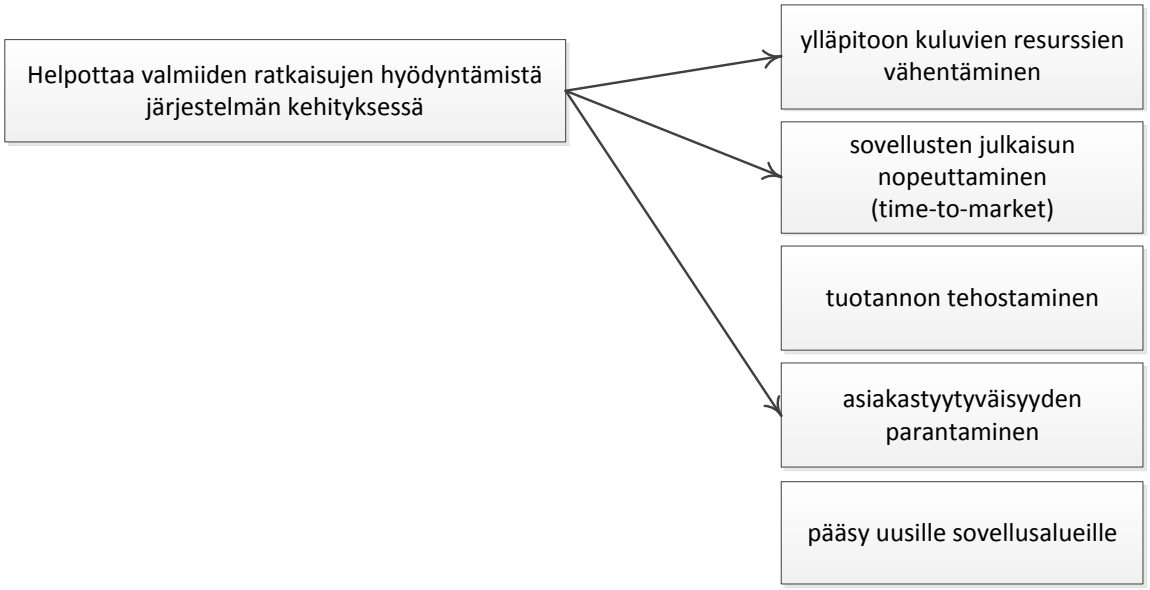
**Helpottaa uusien teknologioiden ja innovaatioiden käyttöönottoa järjestelmän kehityksessä**




Tavoitteen määrittelyssä moderneilla teknologioilla tarkoitetaan sekä järjestelmän kehitysympäristöön, että suoritusympäristöön liittyviä teknologioita. Molempien nykyaikaistamisen arvioitiin tehostavan sovelluskehitystä ja siten nopeuttavan sovellusten julkaisua.

Uusille sovellusalueille pääsyllä tarkoitetaan esimerkiksi järjestelmän kehittämistä käytettäväksi uusilla päätelaitteilla (mm. mobiililaitteet) tai järjestelmän integroimista kolmansien osapuolien tarjoamiin palveluihin (mm. paikkatietopalvelut ja Google Maps). Tarvittavien muutosten toteuttamisen arvioitiin helpottuvan, jos järjestelmän toteutukseen käytettävät teknologiat sekä järjestelmän kehittämiseen käytettävä kehitysympäristö päivitettäisiin mahdollisimman moderneiksi.

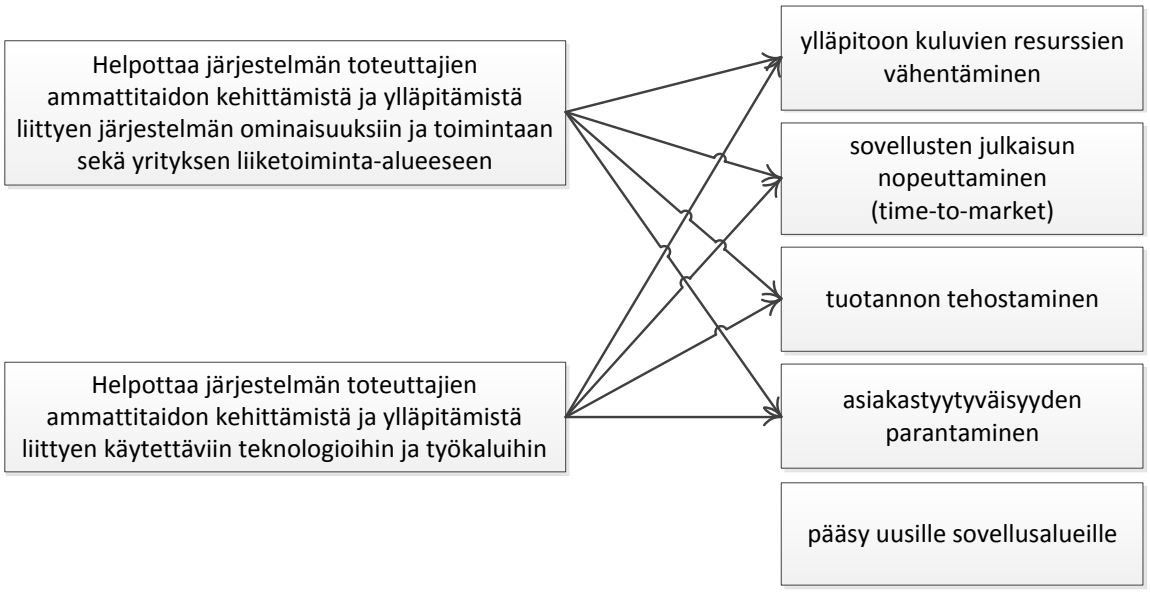
**Helpottaa valmiiden ratkaisujen hyödyntämistä järjestelmän kehityksessä**




Valmiiden ratkaisujen arvioitiin sisältävän vähemmän virheitä kuin yrityksessä kehitettyjen vastaavien komponenttien. Valmiiden komponenttien käytön arvioitiin vähentävän tuotantoon pääsevien virheiden määrää ja siten vähentävän ylläpitoon kuluvia resursseja sekä parantavan asiakastyytyväisyyttä. Sovellusten julkaisu nopeutuu yrityksessä tehtävän kehitystyön vähentymisen myötä.

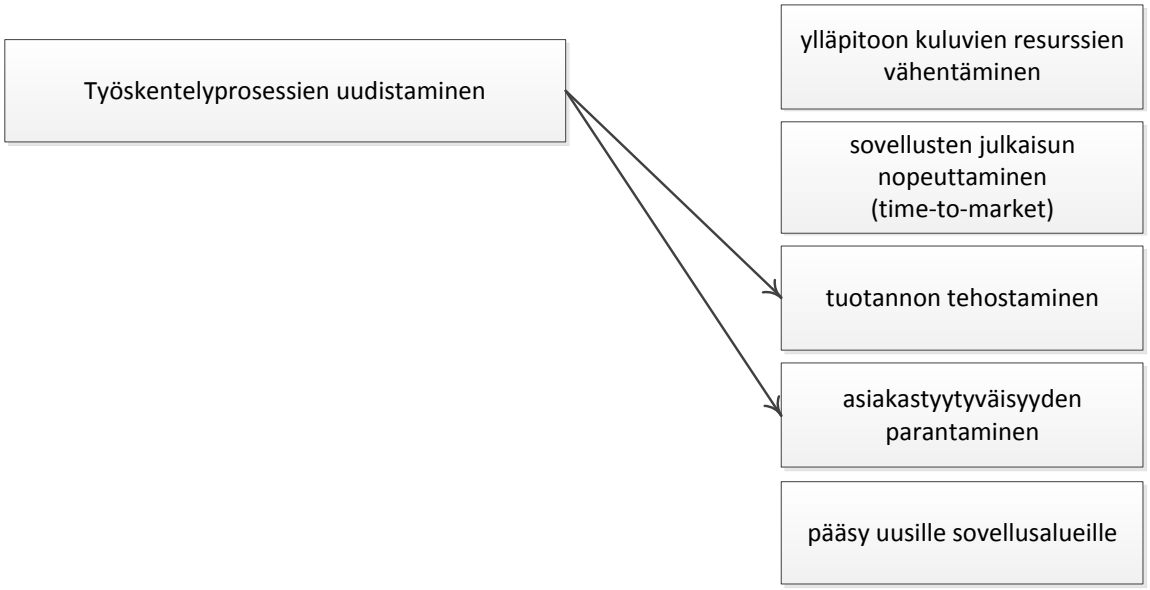
**Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen järjestelmän ominaisuuksiin ja toimintaan sekä yrityksen liiketoiminta-alueeseen**

**Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen käytettäviin teknologioihin ja työkaluihin**



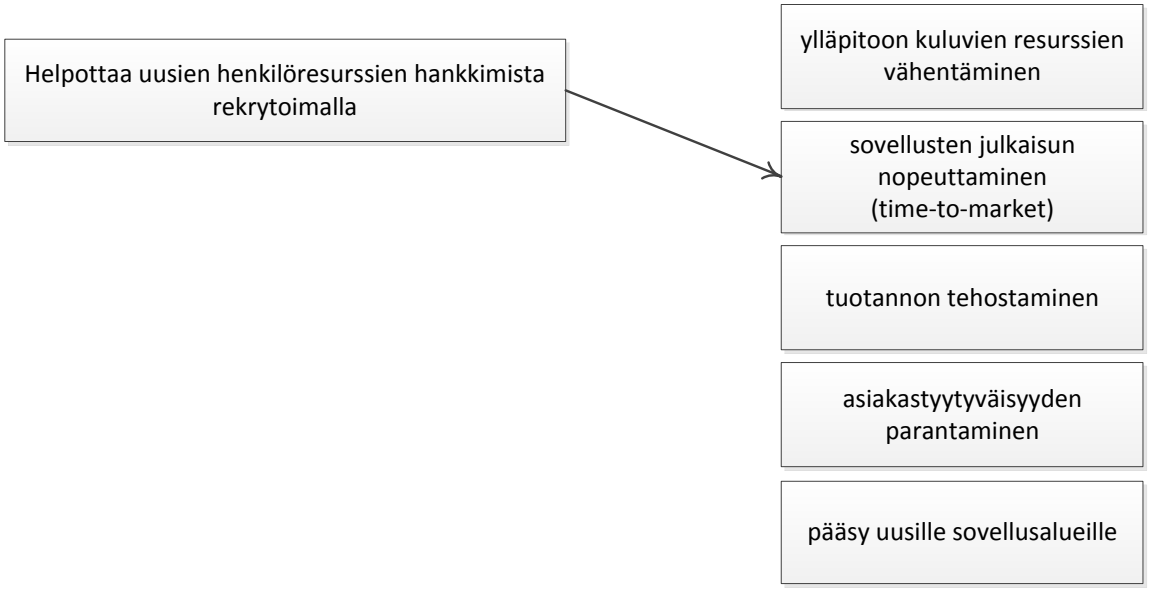

Ohjelmistoa kehittävän henkilöstön ammattitaidon kehittymisen arvioitiin tehostavan tuotantoa, nopeuttavan sovellusten julkaisua ja parantavan kehitettävän järjestelmän laatua. Järjestelmän laadun parantuminen arvioitiin johtavan ylläpitoon kuluvien resurssien vähenemiseen sekä asiakastyytyväisyyden parantumiseen.

**Työskentelyprosessien uudistaminen**



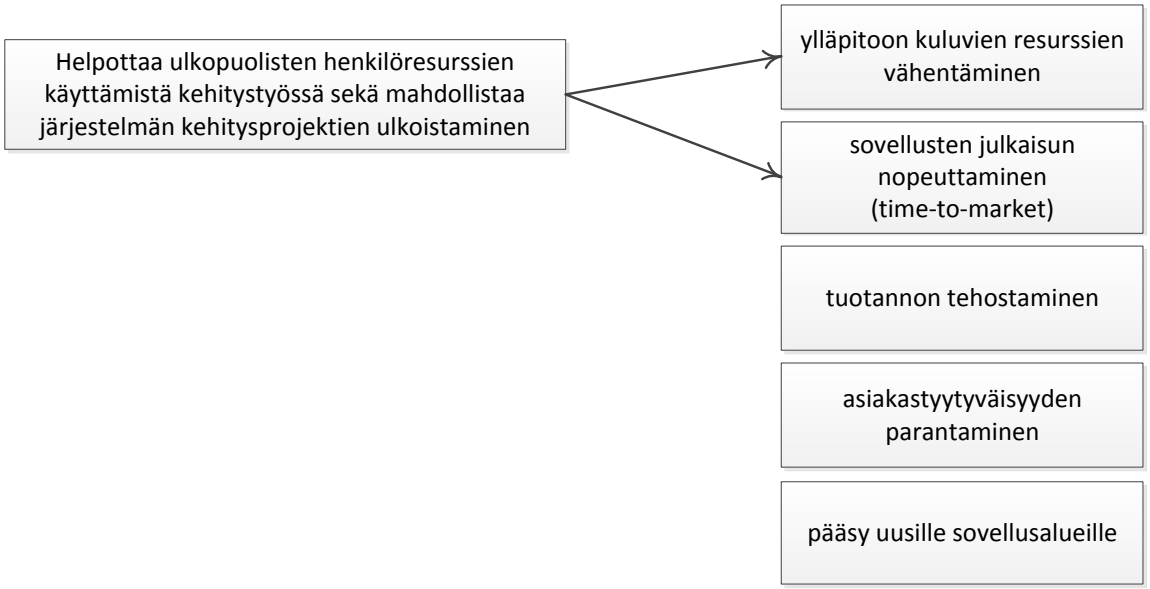

Yrityksen työskentelyprosessien uudistamisen arvioitiin tehostavan ohjelmistotuotantoa. Asiakastyytyväisyyden arvioitiin parantuvan, kun käyttäjätuki ja asiakaspalautteiden käsittely otetaan mukaan uudistettuun työskentelyprosessiin.

**Helpottaa uusien henkilöresurssien hankkimista rekrytoimalla**




Järjestelmää kehittävän henkilöstön lisäämisen arvioitiin nopeuttavan sovelluskehitystä ja sovellusten julkaisua.

**Helpottaa ulkopuolisten henkilöresurssien käyttämistä kehitystyössä sekä mahdollistaa järjestelmän kehitysprojektien ulkoistaminen**




Järjestelmän kehittämiseen käytettävien resurssien lisäämisen arvioitiin nopeuttavan sovelluskehitystä ja sovellusten julkaisua.

Ulkoisten toimijoiden arvioitiin tuottavan vähemmän virheitä sisältäviä komponentteja, jolloin yrityksen omat, ylläpitoon kuluvat resurssit vähenevät.

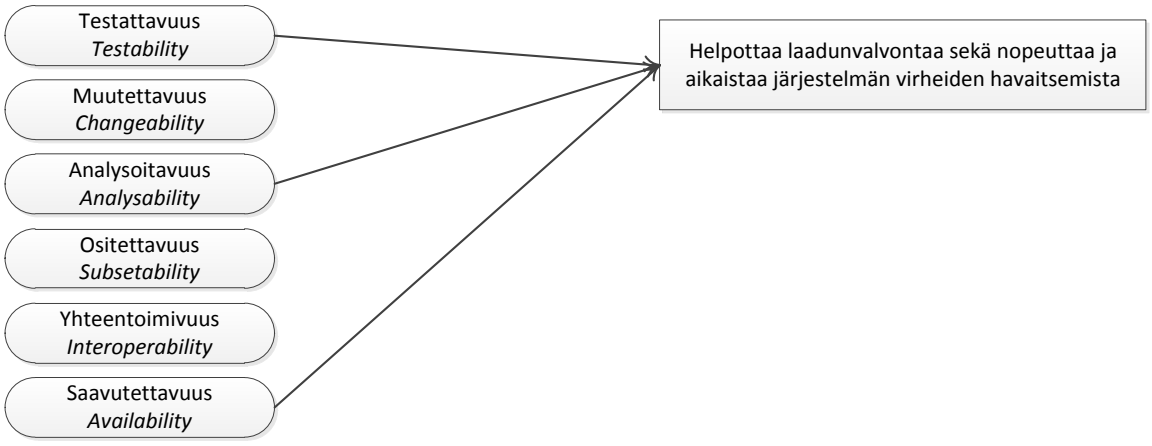
# Liite 2. Laatutekijöiden ja liiketoiminnallisten tavoitteiden väliset suhteet

Esimerkkitapauksessa ohjelmistojärjestelmän tärkeimpien laatutekijöiden valinta perustui järjestelmän liiketoiminnallisten tavoitteiden analysointiin. Analysoinnissa oletettiin, että järjestelmän laadullisiin ominaisuuksiin vaikuttamalla voidaan vaikuttaa myös liiketoiminnallisten tavoitteiden saavuttamisen todennäköisyyteen. Analysoinnin perusteella valitut kuusi järjestelmän kannalta tärkeintä laatutekijää on lueteltu alla tärkeysjärjestyksessä.

1) Testattavuus
2) Muutettavuus
3) Analysoitavuus
4) Ositettavuus
5) Yhteentoimivuus
6) Saavutettavuus

Tässä liitteessä on kuvattu yrityksessä laadittu arvio siitä, mitkä laatutekijät tukevat toteutuessaan minkäkin tarkennetun liiketoiminnallisen tavoitteen saavuttamista.

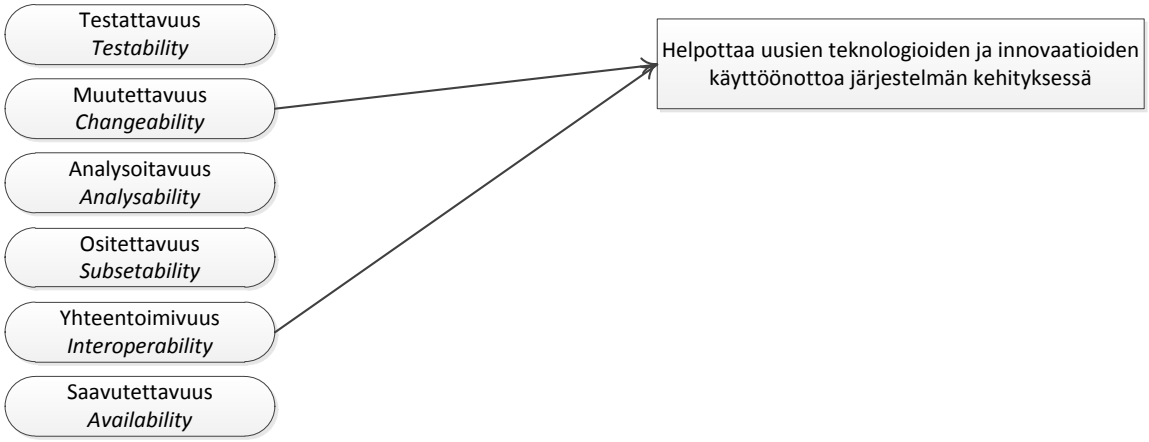
### Helpottaa laadunvalvontaa sekä nopeuttaa ja aikaistaa järjestelmän virheiden havaitsemista




Hyvä testattavuus tukee edellä mainitun liiketoiminnallisen tavoitteen saavuttamista, koska kattavan testauksen avulla voidaan virheitä havaita jo ohjelmiston kehitystyön aikana. Hyvä analysoitavuus mahdollistaa virheiden havaitsemisen toteutetusta koodista

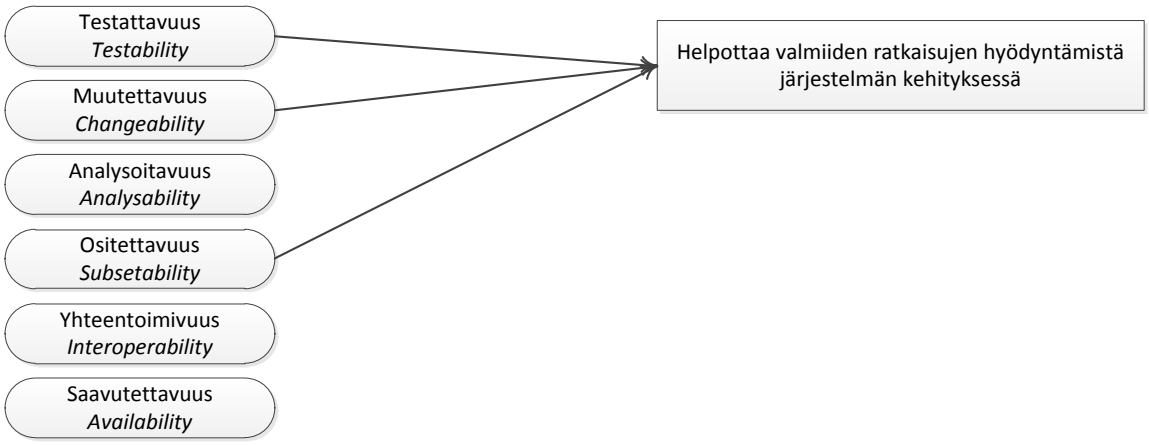
ennen ohjelman suorittamista sekä helpottaa suorituksen aikana havaittujen ongelmien syynä olevien virheiden löytämistä koodista. Saavutettavuutta parantavat arkkitehtuuritason ratkaisut vaikuttavat suoraan edellä mainitun liiketoiminnallisen tavoitteen saavuttamiseen, koska saavutettavuuden saama arvo riippuu mm. virheiden havaitsemiseen kuluvasta ajasta.

**Helpottaa uusien teknologioiden ja innovaatioiden käyttöönottoa järjestelmän kehityksessä**



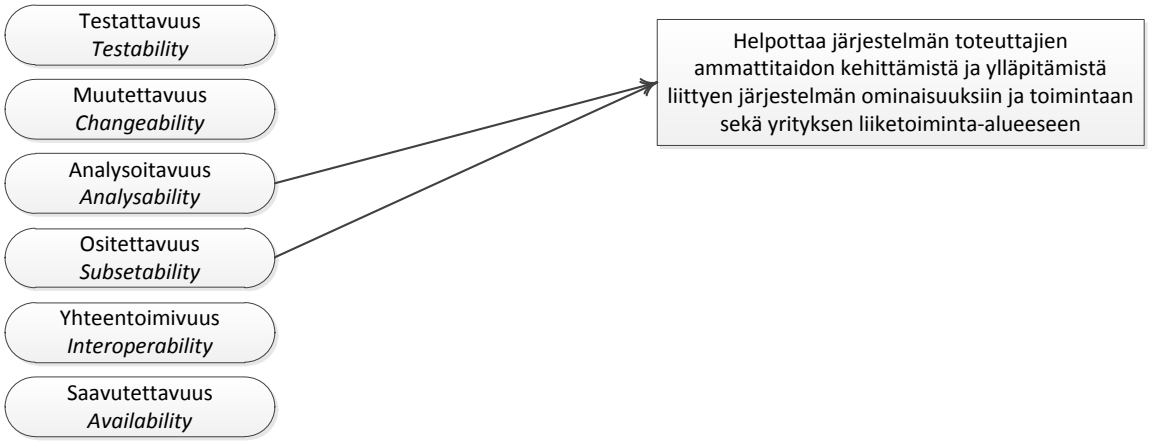

Uusien teknologioiden ja innovaatioiden käyttöönotto vaatii yleensä järjestelmän muuttamista, joten hyvä muutettavuus edesauttaa tämän liiketoiminnallisen tavoitteen saavuttamista. Hyvä yhteentoimivuus vaikuttaa samalla tavalla silloin, kun käyttöön otettavat uudet teknologiat tai innovaatiot on toteutettu erillisinä järjestelminä.

**Helpottaa valmiiden ratkaisujen hyödyntämistä järjestelmän kehityksessä**



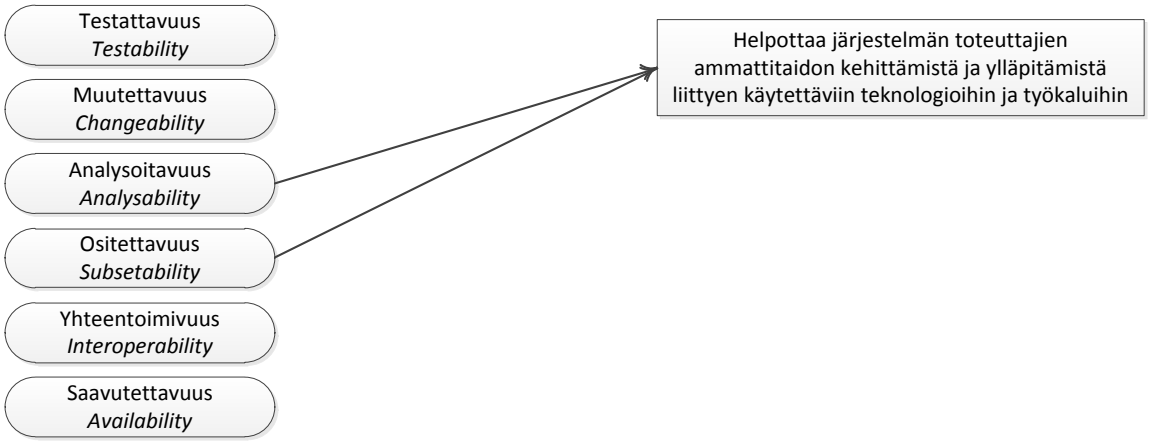

Hyvä testattavuus helpottaa valmiiden ratkaisujen käyttöönottoa koska järjestelmän ja siihen liitettyjen komponenttien oikeellinen toiminta ja integraatio ovat helposti testattavissa. Valmiiden ratkaisujen käyttöönotto ja järjestelmän komponenttien vaihtaminen on sitä helpompaa, mitä paremmalla tasolla järjestelmän muutettavuus on. Järjestelmässä, jonka ositettavuus on hyvä, on komponenttien välillä vähän riippuvuuksia tai riippuvuudet ovat selkeitä ja helposti hallittavissa. Nämä ominaisuudet helpottavat valmiiden ratkaisujen käyttöönottoa järjestelmässä.

**Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen järjestelmän ominaisuuksiin ja toimintaan sekä yrityksen liiketoiminta-alueeseen**



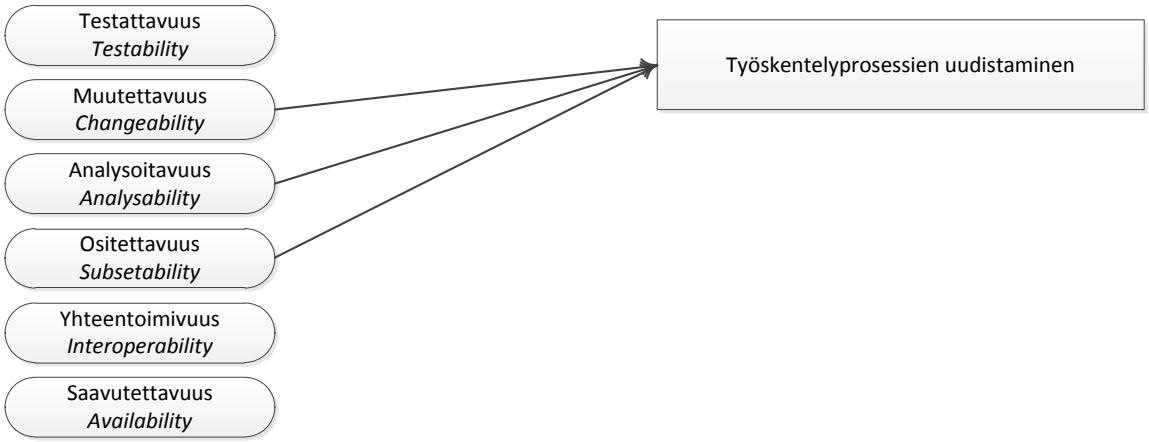

Järjestelmän hyvä analysoitavuus helpottaa sen tekniseen toteutukseen perehtymistä ja eri ominaisuuksien liiketoiminnalliseen tarkoituksen ymmärtämistä. Ositettavuus mahdollistaa järjestelmään perehtymisen selkeä osakokonaisuus kerrallaan.

**Helpottaa järjestelmän toteuttajien ammattitaidon kehittämistä ja ylläpitämistä liittyen käytettäviin teknologioihin ja työkaluihin**



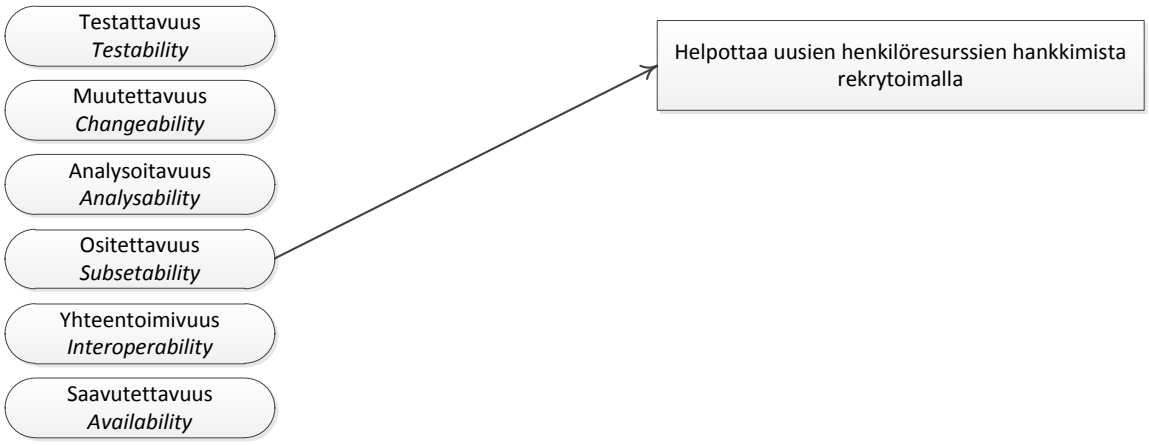

Järjestelmän hyvä analysoitavuus helpottaa sen tekniseen toteutukseen perehtymistä ja auttaa ymmärtämään miksi järjestelmän toteutuksessa käytetään tiettyjä teknologioita ja työkaluja. Ositettavuus mahdollistaa perehtymisen yksi kerrallaan eri teknologioita käyttäviin järjestelmän osakokonaisuuksiin.

**Työskentelyprosessien uudistaminen**



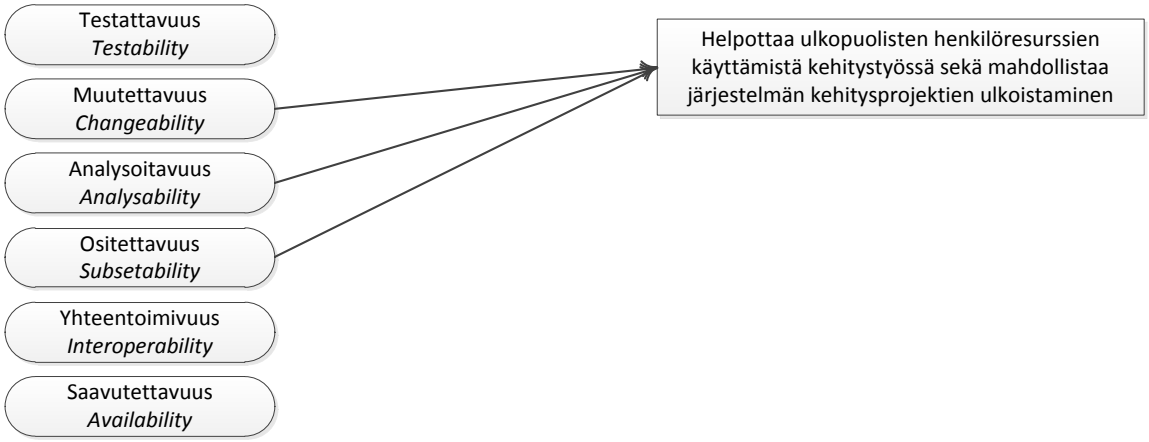

Uudet työskentelyprosessit voivat edellyttää järjestelmältä tietyn tyyppistä rakennetta, jonka toteuttaminen vaatii muutoksia järjestelmään. Hyvä muutettavuus vähentää työskentelyprosessien uudistamisesta järjestelmään kohdistuvia riskejä. Järjestelmän hyvä analysoitavuus ja ositettavuus auttavat järjestelmän rakenteen ymmärtämistä ja vähentävät järjestelmän rakenteesta johtuvia, eri työskentelyprosessien käyttöä vaikeuttavia rajoituksia.

**Helpottaa uusien henkilöresurssien hankkimista rekrytoimalla.**




Hyvä ositettavuus mahdollistaa rekrytoinnin kohdistamisen pelkästään tiettyjen teknologia-alueiden osaajiin.

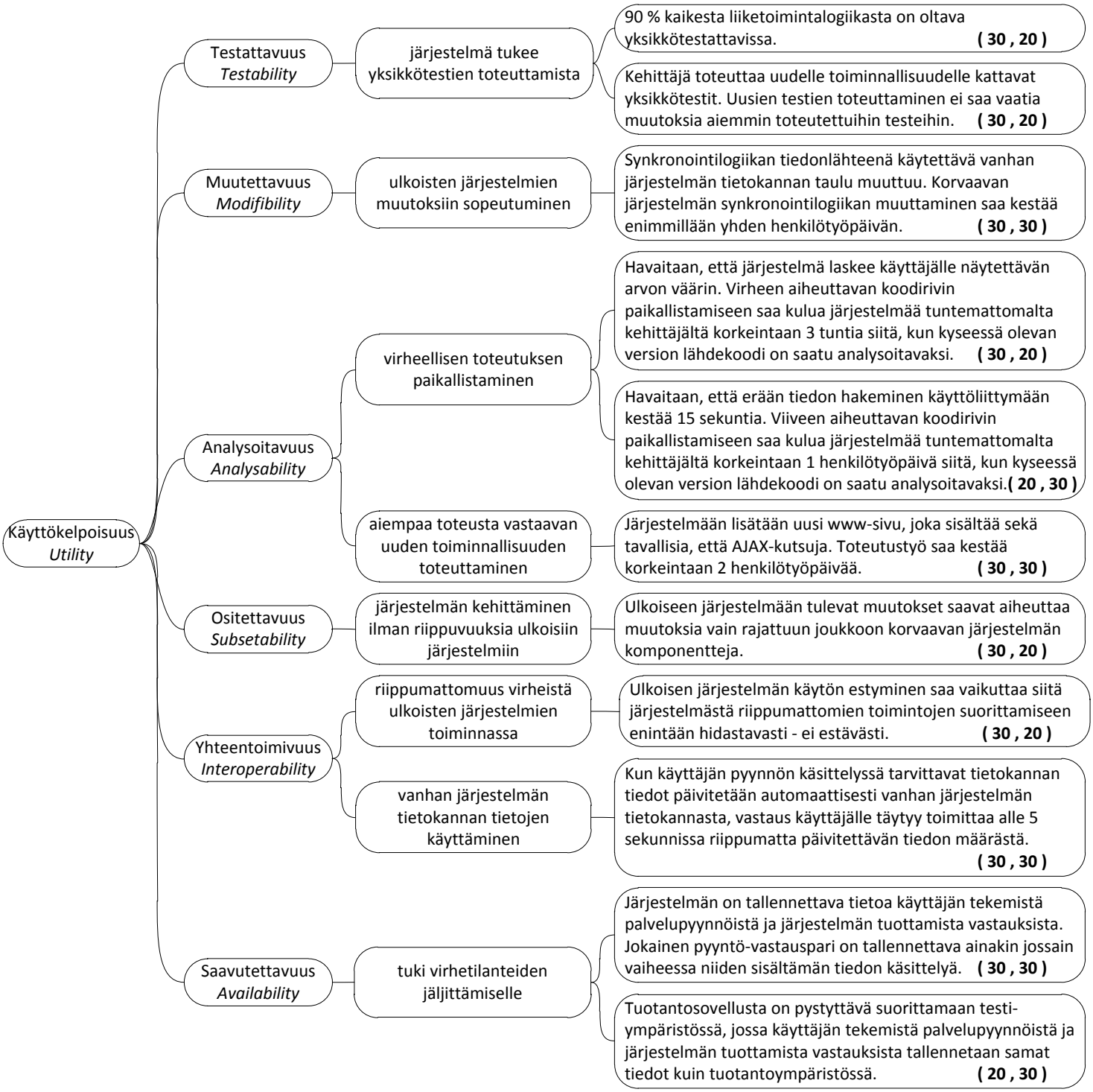
**Helpottaa ulkopuolisten henkilöresurssien käyttämistä kehitystyössä sekä mahdollistaa järjestelmän kehitysprojektien ulkoistaminen**




Järjestelmän eri laatutekijöiden vaikutus ulkoisten resurssien käyttöön riippuu tavasta, jolla ulkoistus toteutetaan. Hyvä muutettavuus ja analysoitavuus helpottavat kaikkea järjestelmään liittyvää kehitystyötä. Tällöin myös järjestelmää aiemmin tuntematon kehittäjä voi omaksua järjestelmän rakenteen nopeasti ja pystyy jo lyhyen perehtymisen jälkeen tehokkaaseen työskentelyyn.

Hyvä ositettavuus tukee kehitysprojektien ulkoistamista erityisesti silloin kun ulkopuoliselta toimittajalta ostetaan järjestelmän selkeä osakokonaisuus.

# Liite 3. Arvioinnissa käytetyt skenaariot laatupuuna



# Liite 4. Kaikki tarkennettuja laatutekijöitä vastaavat skenaariot

**Testattavuus**

| *Järjestelmä tukee yksikkötestien toteuttamista* | | |
|---|---|---|
| 1 | 90 % kaikesta liiketoimintalogiikasta on oltava yksikkötestattavissa. | 30, 20 |
| 2 | Kehittäjä toteuttaa uudelle toiminnallisuudelle kattavat yksikkötestit.<br>Uusien testien toteuttaminen ei saa vaatia muutoksia aiemmin toteutettuihin testeihin. | 30, 20 |
| 3 | Testattavien komponenttien on oltava toteutettavissa sellaisiksi, että niitä voidaan yksikkötestata muusta järjestelmästä riippumattomasti. | 30, 10 |
| 4 | Kehittäjä toteuttaa toiminnallisuuden, joka edellyttää uutta koodia esitystapa- ja liiketoimintalogiikkakerroksiin ja jonka toteuttamiseen kuluu 7 tuntia.<br>Yksikkötestien toteuttaminen 90% symbolikattavuudella saa kestää enimmillään 3 tuntia. | 20, 10 |

| *Järjestelmä tukee skenaariotestien toteuttamista* | | |
|---|---|---|
| 1 | Kehittäjä toteuttaa sovellukseen uuden www-sivun, jolla on 2 kooltaan noin 10 kentän web-lomaketta.<br>Sivun sisältämien käyttöskenaarioiden toimivuuden varmistavien testien kirjoittaminen kestää enimmillään yhden henkilötyöpäivän. | 30, 20 |

| *Järjestelmä ja kehitysympäristö tukevat automaattista testausta* | | |
|---|---|---|
| 1 | Kaikki järjestelmälle laaditut yksikkö- ja skenaariotestit voi suorittaa automaattisesti. | 30, 10 |
| 2 | Kehitysympäristössä oleva järjestelmän versio on pystyttävä siirtämään automaattiseen testaukseen alle 30 minuutissa. | 30, 10 |
| 3 | Tuotannossa oleva järjestelmän versio on pystyttävä siirtämään automaattiseen testaukseen alle tunnissa. | 20, 10 |
| 4 | Kaikki muut testit kuin yksikkö- ja skenaariotestit voi suorittaa automaattisesti. | 10, - |
| 5 | Mikä tahansa järjestelmän versio on pystyttävä siirtämään automaattiseen testaukseen alle 3 tunnissa. | 10, - |

| *Kehitysympäristö tukee rasitustestien suorittamista* | | |
|---|---|---|
| 1 | Järjestelmän testiversiota voidaan kuormittaa tuotantoympäristöä vastaavissa olosuhteissa suorittamalla testiskenaarioita 0 - 500 yhtäaikaista käyttäjää vastaavalla syötteellä. | 10, - |

**Muutettavuus**

| *Muutokset järjestelmän toiminnallisuuteen* | | |
|---|---|---|
| 1 | Mahdollistetaan järjestelmän käyttäminen uudella kielellä. Kaikki kielen mukaan vaihtuva staattinen data on pystyttävä tallentamaan ja hakemaan .resx -tiedostoista. | 20, 20 |
| 2 | WWW-sivun visuaalista ilmettä uudistetaan radikaalisti, tietosisällön pysyessä samana. Muutokset eivät saa heijastua esitystapakerroksen ulkopuolelle. | 20, 10 |

| *Muutokset järjestelmän sisäiseen toteutukseen* | | |
|---|---|---|
| 1 | Liiketoimintalogiikkakerroksen komponentti jaetaan kolmeen erilliseen komponenttiin. Muutoksen toteuttamiseen saa kulua korkeintaan yksi henkilötyöpäivä. | 20, 10 |

| *Muutokset tietorakenteisiin* | | |
|---|---|---|
| 1 | Tietokannan tauluun lisätään uusi kolumni. Pakollisia muutoksia vaativien komponenttien lukumäärä täytyy olla mahdollisimman pieni. | 20, 10 |
| 2 | Tietokannan taulun kolumnin tietotyyppiä muutetaan. Pakollisia muutoksia vaativien komponenttien lukumäärä täytyy olla mahdollisimman pieni. | 20, 10 |
| 3 | Domain-objektiin lisätään uusi persistentti kenttä, joka otetaan käyttöön sekä liiketoimintalogiikka-kerroksessa, että käyttöliittymässä. Kaikkien vaadittujen muutosten toteuttamiseen saa kulua korkeintaan yksi henkilötyöpäivä. | 10, - |

| *Ulkoisten järjestelmien käyttäminen* | | |
|---|---|---|
| 1 | Synkronointilogiikan tiedonlähteenä käytettävä vanhan järjestelmän tietokannan taulu muuttuu. Korvaavan järjestelmän synkronointilogiikan muuttaminen saa kestää enimmillään yhden henkilötyöpäivän. | 30, 30 |
| 2 | Järjestelmän toiminnallisuutta laajennetaan siten, että käyttäjä voi hallita tiettyyn domain-objektiin kuuluvia dokumentteja. Dokumenttien noutoon ja tallentamiseen liittyvän logiikan toteuttamiseen saa kulua korkeintaan 1 henkilötyöpäivä. | 20, 10 |

| *Muutokset suoritusympäristöön* | | |
|---|---|---|
| 1 | Korvaava järjestelmä siirretään kokonaisuudessaan uudelle palvelimelle. Käyttöönotto uudella palvelimella saa kestää enimmillään yhden henkilötyöpäivän. | 10, - |

| *Muutokset käytettyihin kirjastoihin tai toteutusteknologioihin* | | |
|---|---|---|
| 1 | Järjestelmässä siirrytään käyttämään Microsoft ASP.NET MVC-kehyksen uutta versiota, uuden version käyttöönottoon saa kulua enimmillään 3 henkilötyöpäivää. | 10, - |

**Analysoitavuus**

| *Virheellisen tai epäoptimaalisen toteutuksen paikallistaminen* | | |
|---|---|---|
| 1 | Havaitaan, että järjestelmä laskee käyttäjälle näytettävän arvon väärin. Virheen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 3 tuntia siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 30, 20 |
| 2 | Havaitaan, että erään tiedon hakeminen käyttöliittymään kestää 15 sekuntia. Viiveen aiheuttavan koodirivin paikallistamiseen saa kulua järjestelmää tuntemattomalta kehittäjältä korkeintaan 1 henkilötyöpäivä siitä, kun kyseessä olevan version lähdekoodi on saatu analysoitavaksi. | 20, 30 |

| *Aiempaa toteutusta vastaavan uuden toiminnallisuuden toteuttaminen* | | |
|---|---|---|
| 1 | Järjestelmään lisätään uusi www-sivu, joka sisältää sekä tavallisia, että AJAX-kutsuja. Toteutustyö saa kestää korkeintaan 2 henkilötyöpäivää. | 30, 30 |
| 2 | Tiettyyn käyttöskenaarioon lisätään käyttoikeustarkastus. Tarvittavien ehtojen ja tarkastusten toteutustyö saa kestää korkeintaan 3 tuntia. | 30, 10 |
| 3 | Järjestelmään lisätään uusi domain-objekti ja sitä tukeva repository. Toteutustyö saa kestää korkeintaan 3 tuntia. | 30, 10 |

| *Järjestelmän rakenteen ja toteutustavan esittely ja/tai opettaminen* | | |
|---|---|---|
| 1 | Kehittäjä, joka jo tuntee järjestelmän teknologiat, mutta ei korvaavan järjestelmän rakennetta, pystyy 3 viikon opettelun jälkeen osallistumaan tuotantokoodin kehittämiseen, valitulla järjestelmän osa-alueella. | 20, 20 |
| 2 | Järjestelmän tietyn, keskeisen toiminnallisuuden dokumentointi UML-kaavioita käyttämällä saa kestää enimmillään 1 henkilötyöpäivän. | 10, - |

**Ositettavuus**

| *Uuden sovelluslogiikan vaiheittainen toteuttaminen* | | |
|---|---|---|
| 1 | Järjestelmään toteutetaan uuden käyttöskenaarion vaatima sovelluslogiikka. Esitystapakerroksen komponentit täytyy pystyä toteuttamaan ja testaamaan riippumatta liiketoimintalogiikkakerroksen toteutusaikataulusta. | 30, 10 |
| 2 | Järjestelmään kuuluvaa käyttöskenaariota on pystyttävä esittelemään, vaikka muihin käyttöskenaarioihin kuuluvat sivut ja niitä tukeva logiikka ei olisi vielä valmista. | 20, 10 |
| 3 | Järjestelmään on lisätty väliaikaista sovelluslogiikkaa asiakasdemoa varten. Tiettyyn käyttöskenaarioon liittyvän demototeutuksen poistamiseen saa kulua korkeintaan 1 tunti. | 20, 10 |
| 4 | Asiakkaalle esitellään uuden toiminnallisuuden ulkoasua. Esitystapakerrokseen toteutettavan, liiketoimintalogiikkakerroksen toimintaa jäljittelevän demototeutuksen laatimiseen saa kulua enintään yksi henkilötyöpäivä. | 10, - |

| *Vanhan sovelluslogiikan vaiheittainen muuttaminen* | | |
|---|---|---|
| 1 | Uuden, korvaavan sovelluslogiikan toteuttamisen tulee olla mahdollista ilman että vanha toteutus jouduttaisiin ensin poistamaan. | 30, 10 |
| 2 | Liiketoimintalogiikkakerroksen komponenttien vaihtaminen toiminnaltaan vastaaviin, mutta toteutukseltaan erilaisiin, täytyy olla mahdollista ilman pakollisia muutoksia komponenttien käyttäjien toteutukseen. | 30, 10 |

| *Järjestelmän kehittäminen ilman riippuvuuksia ulkoisiin järjestelmiin* | | |
|---|---|---|
| 1 | Ulkoiseen järjestelmään tulevat muutokset saavat aiheuttaa muutoksia vain rajattuun joukkoon korvaavan järjestelmän komponentteja. | 30, 20 |

**Yhteentoimivuus**

| *Riippumattomuus virheistä ulkoisten järjestelmien toiminnassa* | | |
|---|---|---|
| 1 | Ulkoisen järjestelmän käytön estyminen ei saa johtaa tunnistamattomien tai hallitsemattomien vikojen esiintymiseen järjestelmässä. | 30, 20 |
| 2 | Ulkoisen järjestelmän käytön estyminen saa vaikuttaa ko. järjestelmästä riippumattomien toimintojen suorittamiseen enintään hidastavasti - ei estävästi. | 20, 20 |

| *Vanhan järjestelmän tietokannan tietojen käyttäminen* | | |
|---|---|---|
| 1 | Kun käyttäjän pyynnön käsittelyssä tarvittavat tietokannan tiedot päivitetään automaattisesti vanhan järjestelmän tietokannasta, vastaus käyttäjälle täytyy toimittaa alle 5 sekunnissa riippumatta päivitettävän tiedon määrästä. | 30, 30 |
| 2 | Vanhan järjestelmän tietokannan tietojen käyttö korvaavasta järjestelmässä ei saa estää vanhan järjestelmän toimintaa riippumatta korvaavan järjestelmän toiminnan oikeellisuudesta. | 30, 10 |

| *Dokumenttipalvelimen käyttäminen* | | |
|---|---|---|
| 1 | Tiettyyn rakennukseen, kiinteistöpalveluun tai työtehtävään liittyvät dokumentit noudetaan dokumenttipalvelimelta, dokumenttien nouto saa kestää korkeintaan 5 sekuntia. | 20, 10 |

| *Vanhan järjestelmän käyttäjätunnistukseen tukeutuminen* | | |
|---|---|---|
| 1 | Korvaavan järjestelmän on tarjottava käyttäjille vanhan järjestelmän käyttäjätunnistukseen perustuva, automaattinen sisäänkirjautuminen riippumatta siitä, onko käyttäjä aiemmin käyttänyt korvaavaa järjestelmää. | 30, 10 |

**Saavutettavuus**

| *Tuki virhetilanteiden havaitsemiselle* | | |
|---|---|---|
| 1 | Järjestelmän käytön estävistä virhetilanteista on informoitava järjestelmän käyttäjiä, jotka yrittävät käyttää virheen vaikutuspiirissä olevia toimintoja. | 30, 10 |
| 2 | Palvelimen ja järjestelmän kykyä vastata http-pyyntöön on seurattava järjestelmästä riippumattomalta palvelimelta ja havaituista virheistä on toimitettava tieto määriteltyjen henkilöiden sähköpostiin virheen havaitsemisen yhteydessä. | 20, 10 |
| 3 | Järjestelmässä havaitusta, käyttöskenaarion suorittamisen estäneestä ohjelmistovirheestä on toimitettava tieto määriteltyjen henkilöiden sähköpostiin välittömästi virheen havaitsemisen jälkeen. | 20, 10 |

| *Tuki virhetilanteiden jäljittämiselle* | | |
|---|---|---|
| 1 | Järjestelmän on tallennettava tietoa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista. Jokainen pyyntö-vastauspari on tallennettava ainakin jossain vaiheessa niiden sisältämän tiedon käsittelyä varten. | 30, 30 |
| 2 | Tuotantosovellusta on pystyttävä suorittamaan testiympäristössä, jossa käyttäjän tekemistä palvelupyynnöistä ja järjestelmän tuottamista vastauksista tallennetaan samat tiedot kuin tuotantoympäristössä. | 20, 30 |

| *Tuki virhetilanteiden korjaamiselle* | | |
|---|---|---|
| 1 | Sovelluksen korjatun version julkaisu tuotantoon saa kestää enimmillään yhden henkilötyötunnin. | 20, 10 |
| 2 | Tuotantotietokannan muutosten toteuttaminen on oltava mahdollista ilman tuotantosovelluksen uuden version julkaisemista. | 10, - |

| *Virheiden vaikutusalueen rajaaminen* | | |
|---|---|---|
| 1 | Tietyn käyttöskenaarion suorittamisen estävä ohjelmistovirhe ei saa vaikuttaa järjestelmään siten, että sellaisten käyttöskenaarioiden suorittaminen estyy, joihin virhe ei suoraan vaikuta. | 30, 10 |